山のあなたの空遠く
ー構造化プログラミングー

岐阜経済大学　経営学部 経営情報学科　井戸　伸彦

来歴：

0．0版　２００３年4月

1．0版　２００４年4月：(2)に追記

2．0版　２００５年11月：(4)(5)に追記

スライドの構成

はじめに

(1) 導入

(2) PADで描くプログラム

(3) ツール“PadSimulator”

(4) だんだん詳しく設計する

(5) 構造化とgoto文

(6) 階層構成

(7) 制御構造を壊すもの

おわりに

はじめに

- 受講者は、初歩のプログラミングの経験があるものとしています。
- 説明は直感的な分かりやすさを重視し、厳密には不正確な言い方もしています。
- 文中、“オブジェクト指向”との言葉が何回か現れますが、これについては、別講座で取り上げます。
- ツールとして、PadSimulatorを用います。“PadSimulator”は、次のサイトからダウンロードできる、フリーソフトウェアです。
  - http://www.vector.co.jp/soft/win95/prog/se235608.html
- プログラミング言語としては、Ｊａｖａを扱います。次の資料を、本文中で「Ｊａｖａスライド」と参照しています。
  - シャバドゥビ、ジャバ　ーJava覚書ー
    (http://www.gifu-keizai.ac.jp/~ido/doc/java/java_text.pdf)
- Ｊａｖａの実行環境としては、eclipseを前提としてますが、本スライド中で特にeclipseに関係する部分はありません。

(1．1) まずは、思考実験

■A君の午前についての、次の記述を読んでください。

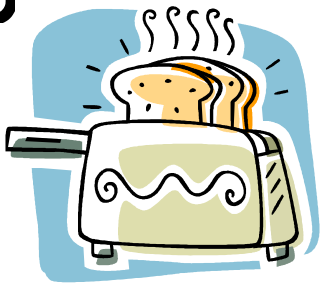
- A君は目覚めると、すぐにシャワーに入る。
- シャワーを出た後、冷蔵庫の中身を調べる。
  ご飯があればご飯を食べる。
  ご飯が無ければ読書をする。
- 食事の後、晴れていれば散歩に出かける。
  晴れていない日は、家で読書をする。
- 散歩から帰ると、音楽を聴く。
- 音楽を聴いて気分が良くなれば、読書をする。そうでなければ、散歩をする。
- 読書が終わると午前は終了、これから昼飯！

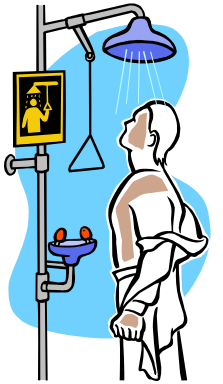
■A君の午前について、想像がつきましたか？

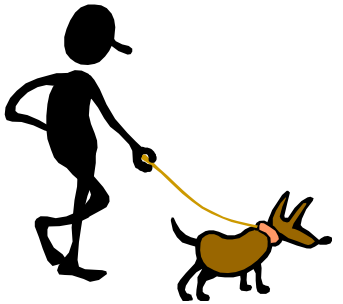
(1．2) 少し分かりやすく。。。

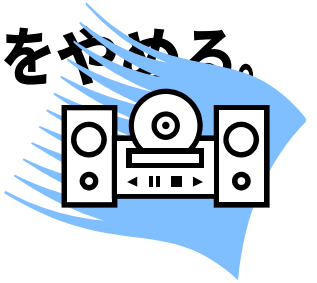
■前のスライドと同じことを、次のように書いてみます。

- シャワーに入る。
- 冷蔵庫にご飯はあるか？
  - ◆Yes
    - □ご飯を食べる。
    - □晴れているか？
      - ≫Yes
        - ＋次の動作を繰り返す。
          - ・散歩をする。
          - ・音楽を聴く。
          - ・気分が良いのであれば、繰り返しをやめる。
- 読書をする。
- 午前は終了、これから昼飯！









■このほうが理解しやすくはありませんか？

(1．3) 構造化プログラミング

■(1.1)と(1.2)との違いを見ると、手順を書くときには、良い書き方があることが想像されると思います。

■構造化プログラミングは、

「どうしたら良いプログラムを書くことが出来るか？」

という問いに答えるもののひとつです。

■(1.1)と(1.2)との違いが“構造化プログラミング”という訳ではありませんが、(1.2)が良いというのは、何故でしょうか？

- 正しく動く
  - ◆(1.1)、(1.2)とも同じ動きをします。両方とも正しく動いています。
- 正しく動くことが、分かりやすくなっている。
  - ◆この点で、(1.1)と(1.2)との違いが生じているよ

■すなわち、正しく動くだけでなく、

それが確認しやすいプログラムが

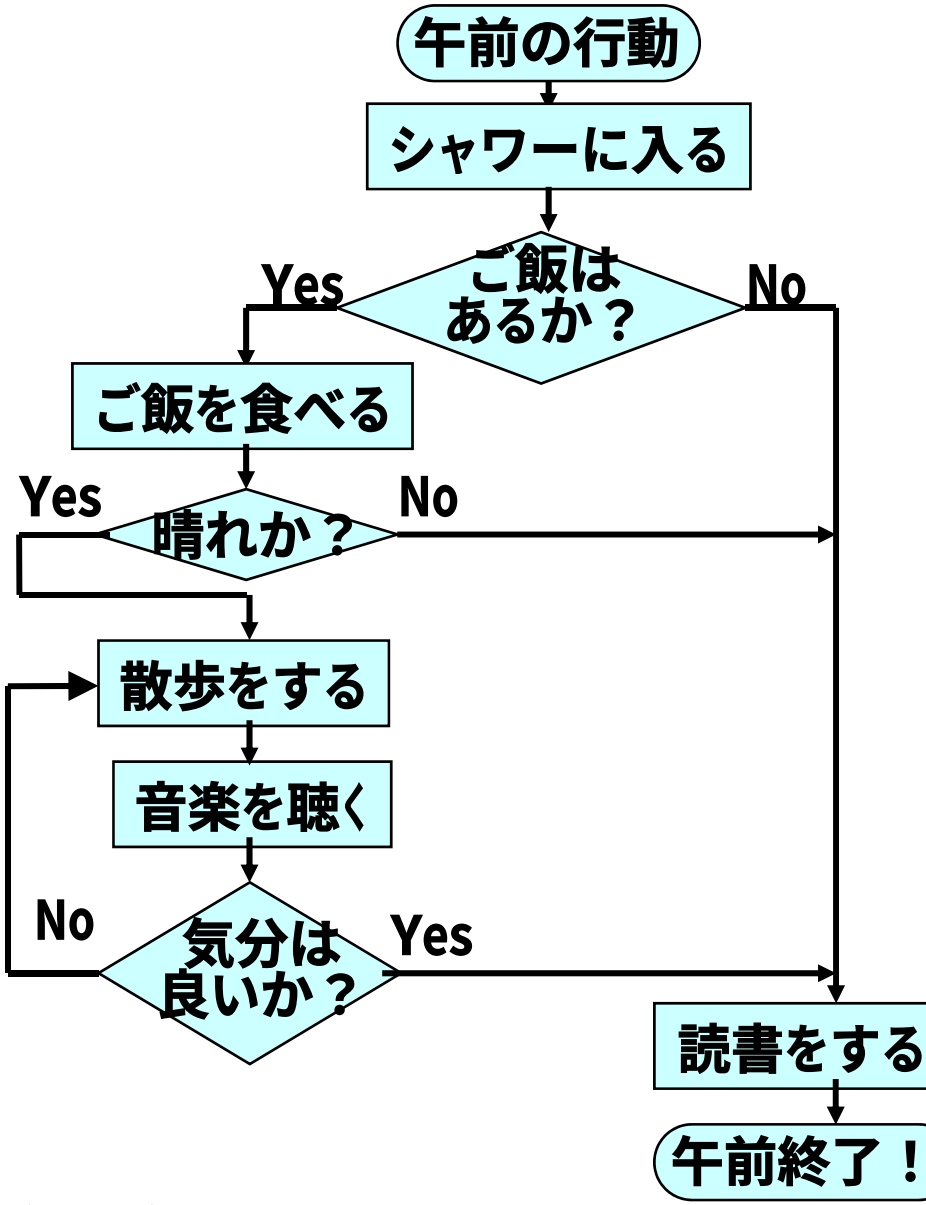
良いプログラムということになります。

(2．1) フローチャート

■文章よりも、図にしたほうが分かりやすいということは、よくあることです(実際、ソフトウェア設計のための様々な図があります)。

■スライド(1.1)(1.2)のA君の動作を、フローチャートという図にしてみました。

■確かに分かりやすくなった気がするのですが、どこか(1.1)のような雰囲気があります。

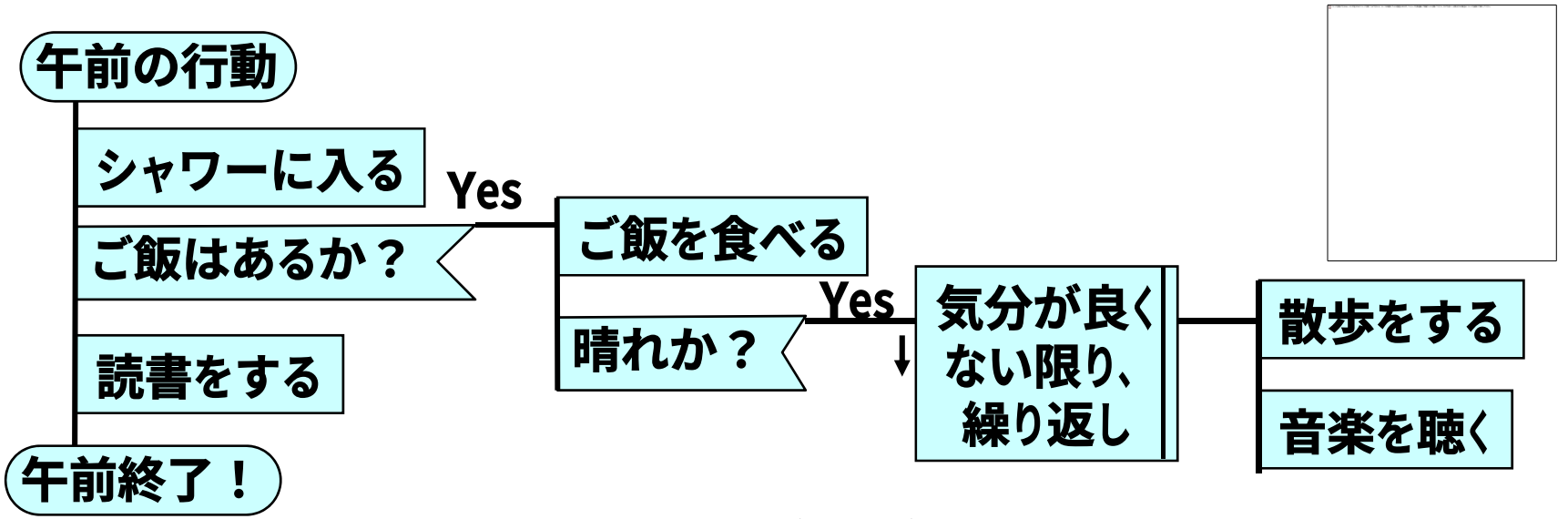


(2．2) 構造化プログラミングにおける記法

■下図は、スライド(1.1)(1.2)のA君の動作を、PADという記法 (図) にしてみたものです。今度は、(1.2)風になっています。

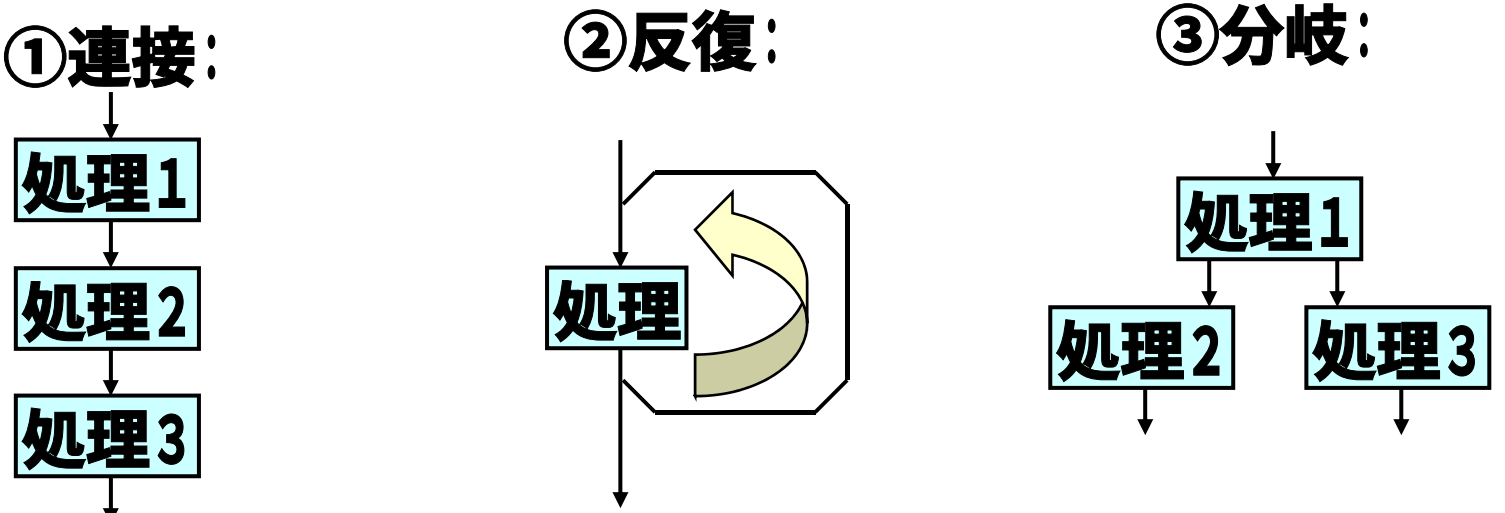
■現在、商用プログラムの設計 (詳細設計) では、フローチャートではなく、(1.2)風のPADのような記法が用いられます。

■(1.2)風の記法とは、どのような記法なのでしょうか？



(2．3) 3つの基本単位

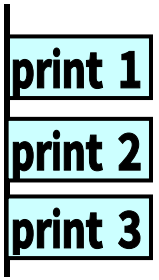
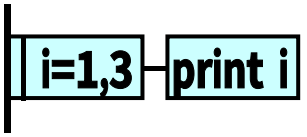
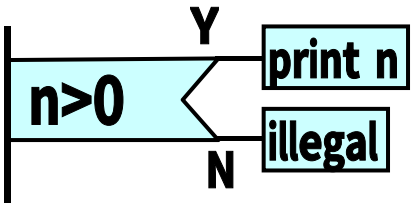
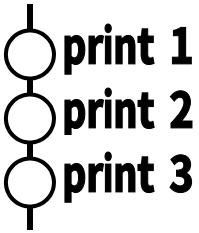
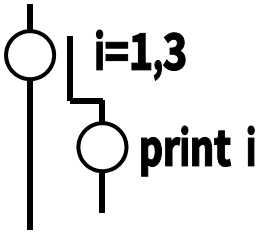
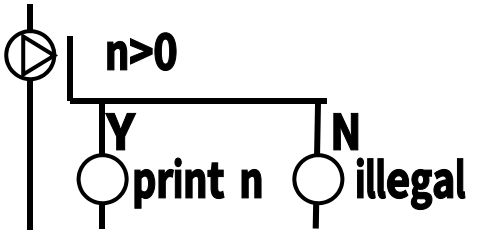
■構造化プログラミングで用いられる記法では、次の3つを組み合わせてプログラムを記します。



■皆さんが勉強したことのあるプログラミング言語 (C、VB、Perl、etc.) は、多くの場合、手続き型とよばれる種類に属します。手続き型であれば、プログラミング言語によらず、この3つをどのように組み合わせるかにより、プログラムを書くことが出来ます。

■次のスライドにて、組み合わせ方を記す記法を紹介します。

(2．4) さまざまな記法

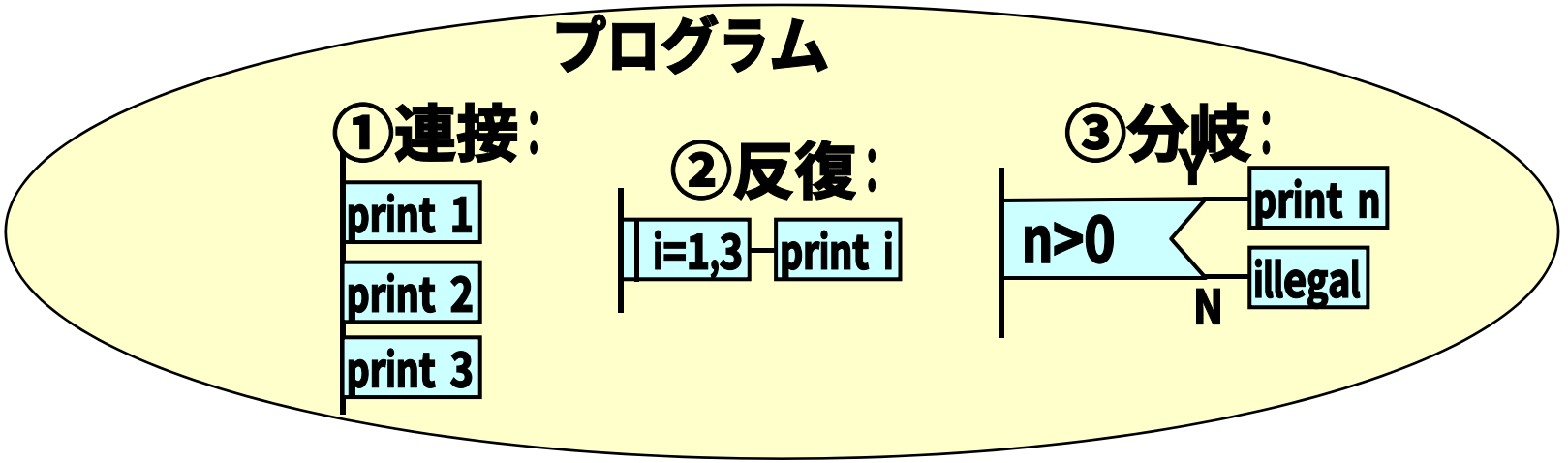
	①連接：	②反復：	③分岐：
PAD			
HCP チャート			
IPC	： 2.1 print 1 2.2 print 2 2.3 print 3 同じネスト（＝書き始めの位置と番号の桁）の続きの番号は、順次実行。	： 2.1 i=1,3にて繰り返し 2.1.1 print i 繰り返しの部分は、ネストをひとつ下げる。	2.1 n>0かを判定 2.1.1 n>0の場合 (1)print n 2.1.2 そうでない場合 (1)illegal 判定条件はひとつ、実行文書はもうひとつ、ネストを下げる。

(2．5) PAD

■本講座では、日立製作所で開発された、PAD (Program Analysis Diagram) という記法を用いることにしています。PADは、構造化プログラミングにて用いられる記法のうち、最も一般的なもののひとつです。

■PADの描き方は難しくありません。その規則は、前のスライドに記した①連接、②反復、③分岐を組み合わせることに尽きます。

■繰り返し言いますが、あらゆる処理 (プログラム) は、①連接、②反復、③分岐の3つを組み合わせることで実現出来ます。



(2．5．1) 連接

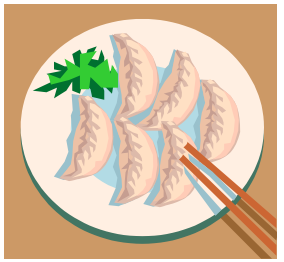
■連接では、順次実行される処理を、次の形式で記します。

最初に実行される処理
次に実行される処理
その次に実行される処理

■例：

ラーメンを食べる
餃子を食べる
ビールを飲む

準備運動をする
サッカーの試合をする
シャワーを浴びる



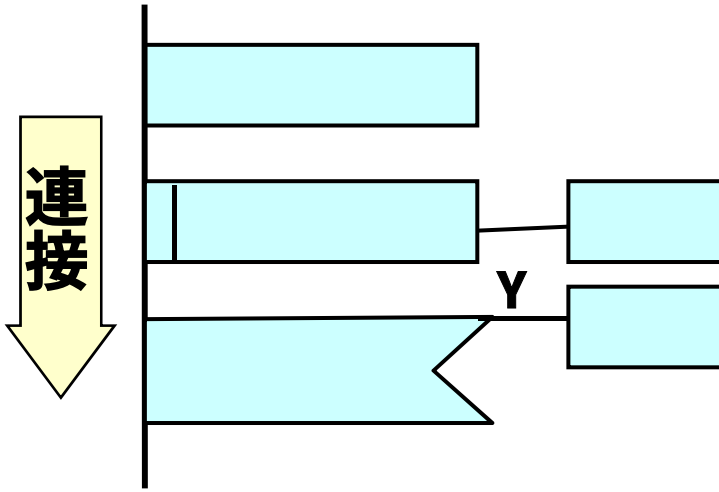
(2．5．2) 処理を記す四角

■四角 (□) の中に処理を書くというのは、ＰＡＤでの基本的な約束ごとです。

処理を書く

■この四角 (□) は、「反復」、「分岐」の要素とは区別されます。すなわち、反復や分岐を四角 (□) に書くことはしません。

■前スライド(2.5.1)では、連接される (順次実行される) ものとして、四角 (□) に書かれた処理をあげていますが、「反復」、「分岐」も同様に連接されます。



(2．5．3) 反復

■反復では、「繰り返しを行う条件」と、「繰り返し実行される内容」とを、次の形式で記します。

繰り返しを行う条件	繰り返し実行される内容

■例：

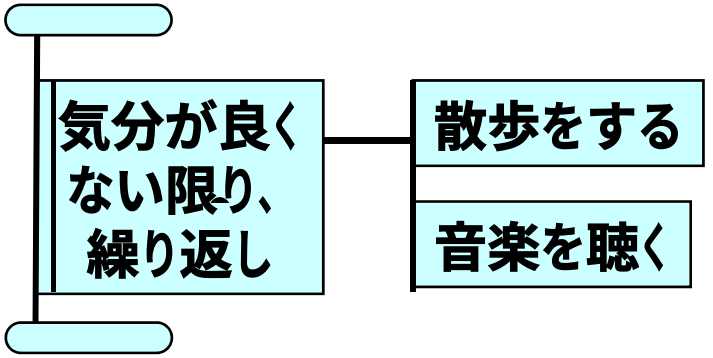
3回繰り返す	屈伸

塩が足りない	塩を足す
	かき混ぜる

(2．5．4) 前判定･後判定

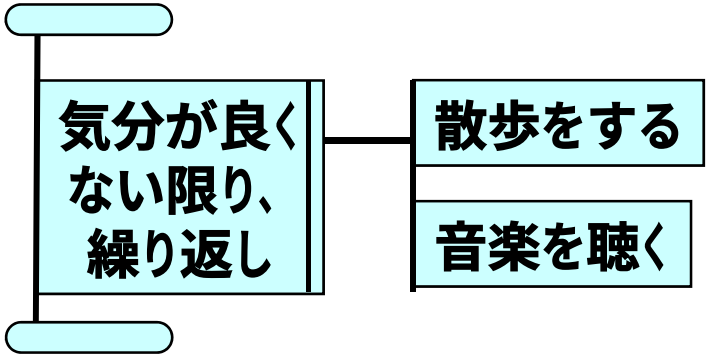
■反復については、次の2種類があることを覚えておきましょう。

＜前判定 □ ＞

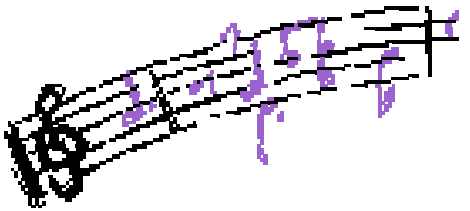


⇒最初から気分が良ければ、散歩･音楽は一回もしない。

＜後判定 □ ＞

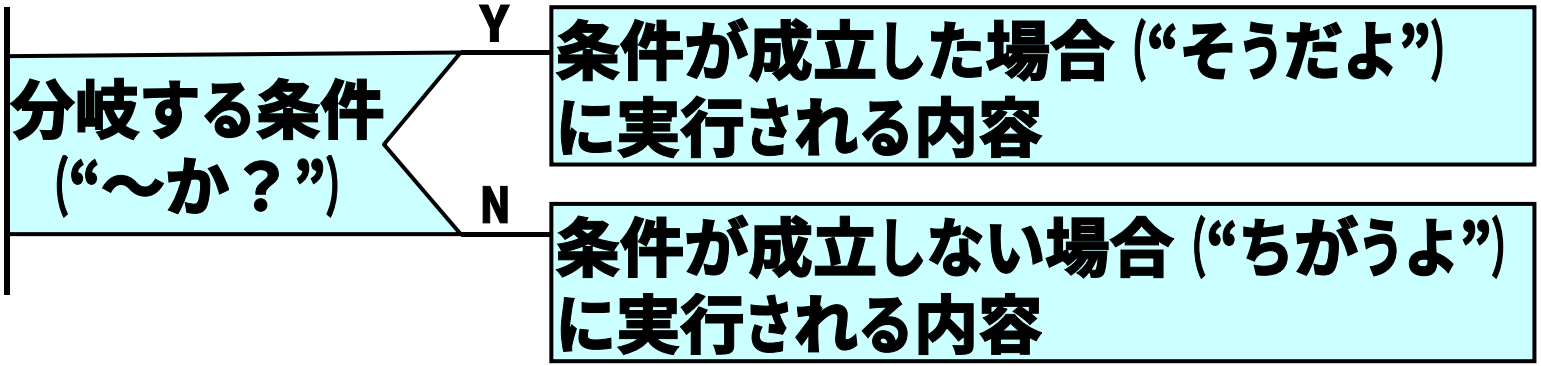


⇒最初から気分が良くても、一回は散歩･音楽をする。

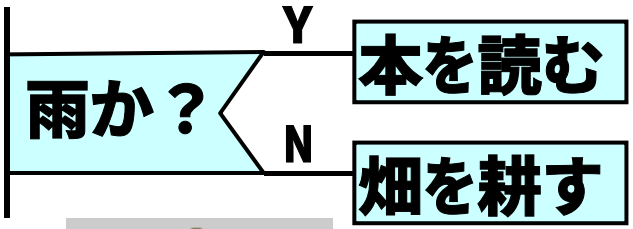


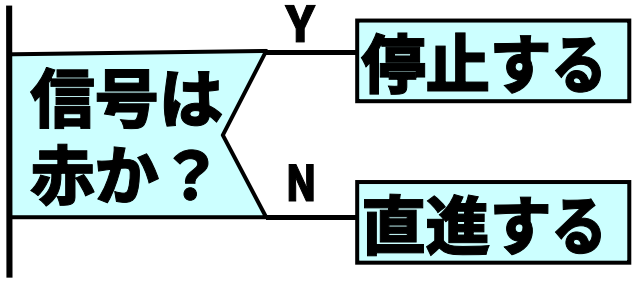
(2．5．5) 分岐

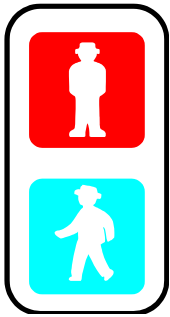
■分岐では、「分岐する条件 (“～か？”というような質問)」と、「条件が成立した場合に実行される内容」、および、「条件が成立しない場合に実行される内容」を、次の形式で記します。



■例:

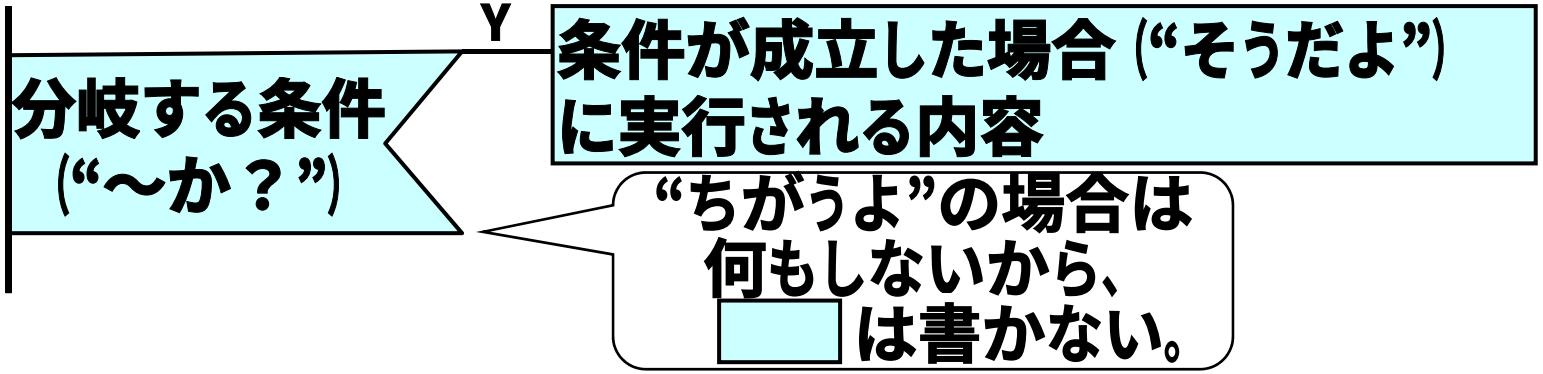




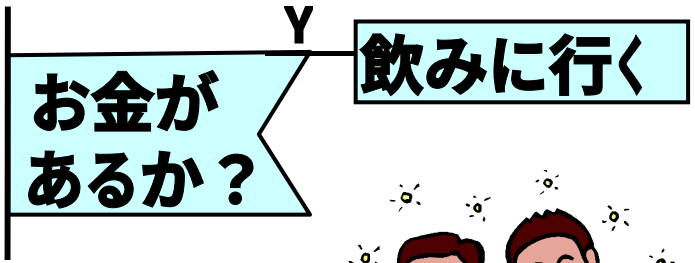


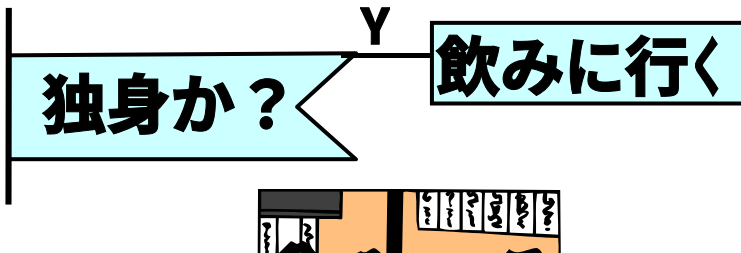
(2．5．6) 片方だけの分岐

■分岐にて、条件が成立しない場合は何もしないならば、“N”に対応する処理の四角は書きません。



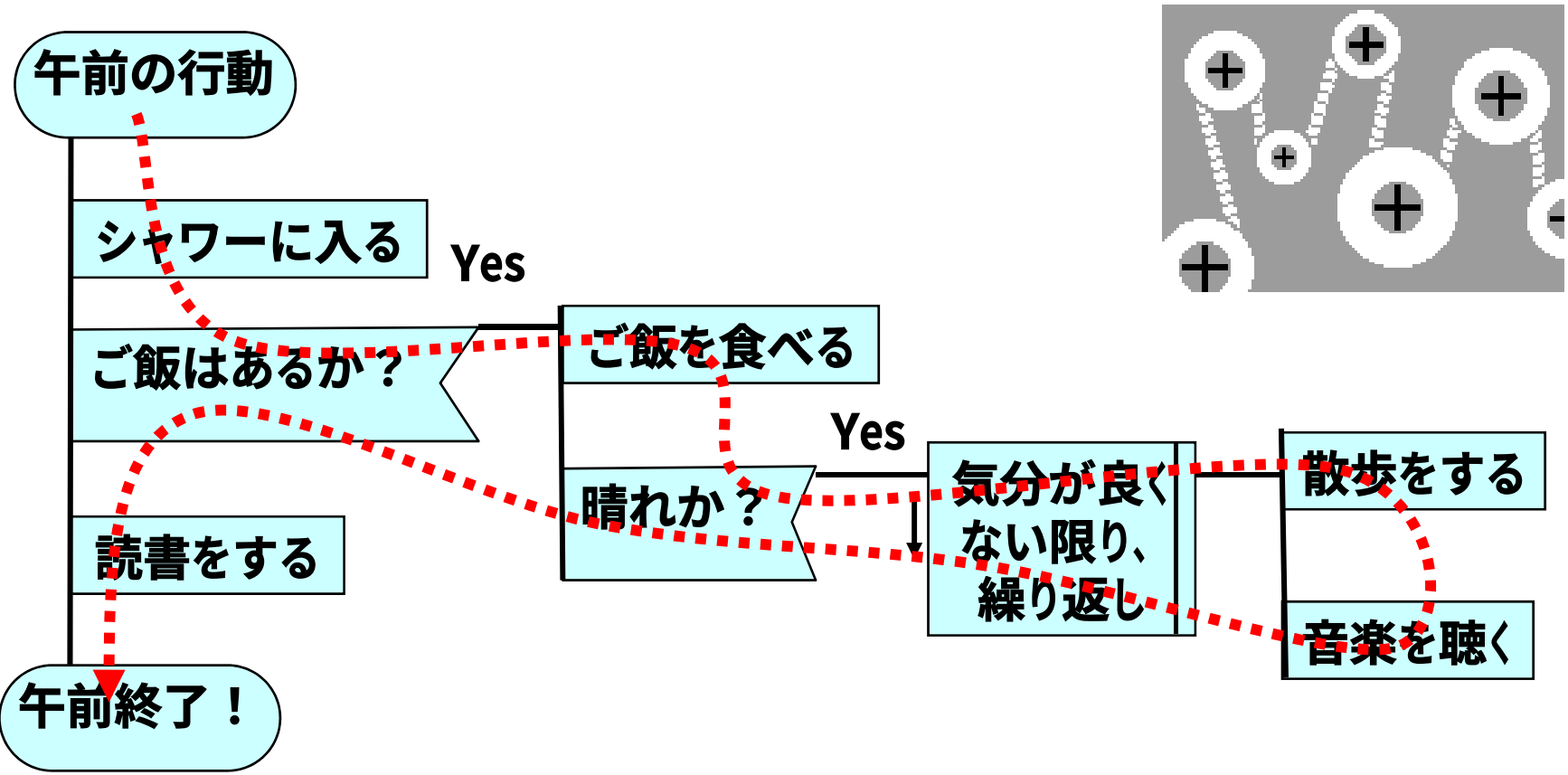
■例：





(2．6) 処理の流れ

■PADでの処理の流れについて、その気分を下の図に示します (ツリー構造を深さ優先で辿ることになります)。

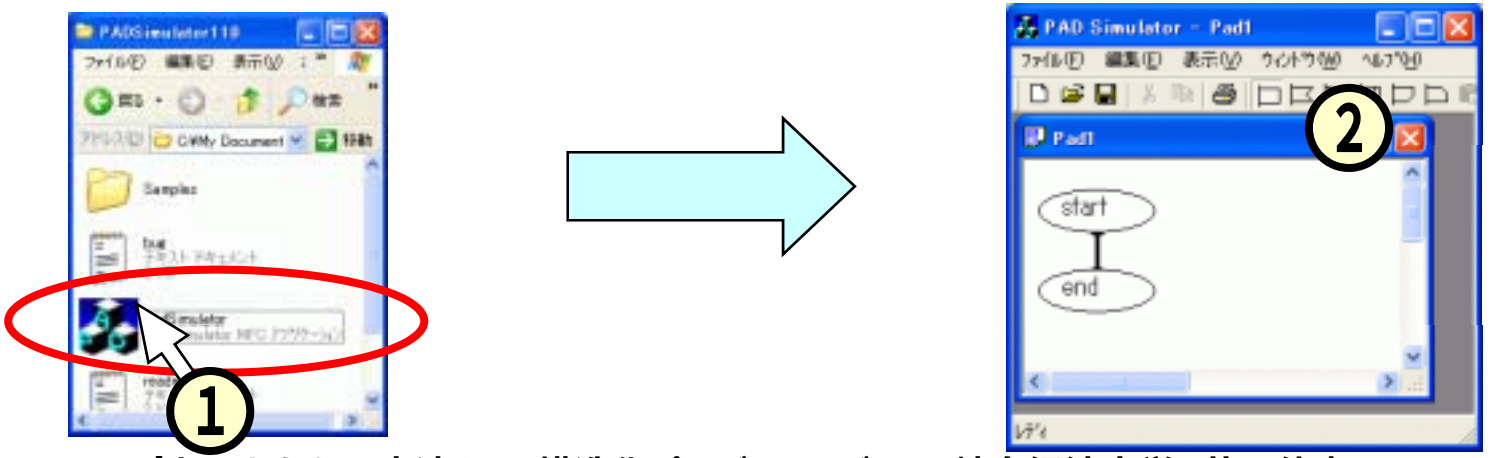


(3) PadSimulator

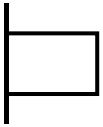
■PADを描く際、本講座では“PadSimulator”というツールを使います。

■ファイルはWindowsでの実行形式 (.exe) になっており、これをダブルクリック (①) すると、ツールが立ち上がります (②)。

(大学内では、[スタート]-[すべてのプログラム]-[アプリケーション]-[PadSimulator]から起動出来ます。)

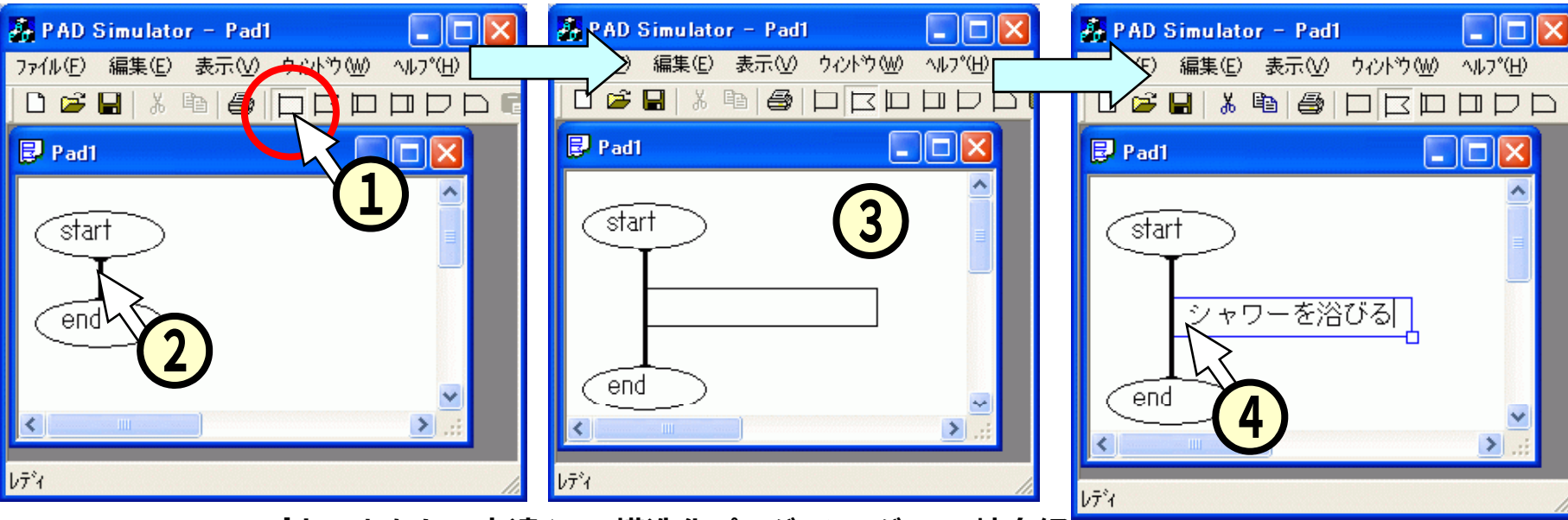


(3．1) 要素の挿入

■挿入したい要素 (右図の場合は、連接 ) を、メニューから選択してクリック (①) します。

■挿入したい場所をクリック (②) すると、その場所に選択した要素が挿入されます (③)。

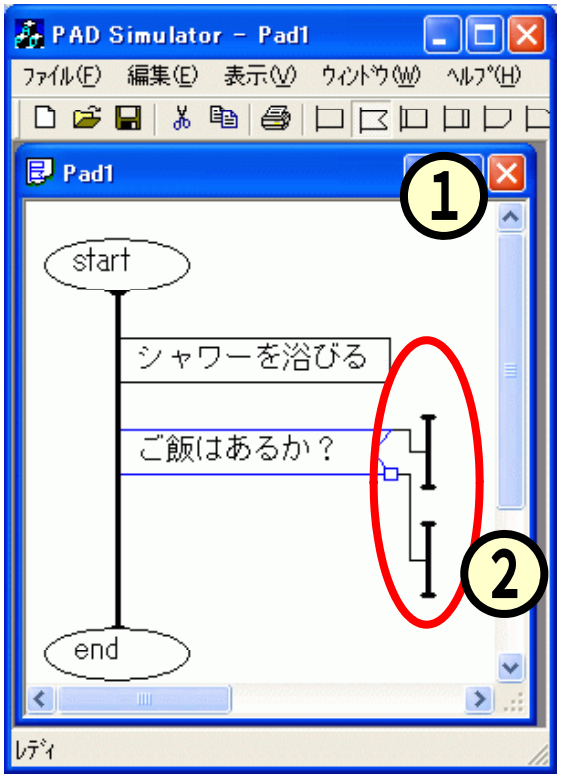
■要素内をクリック (④) すると、文字が入力できるようになります。他の要素も同様に挿入できます。

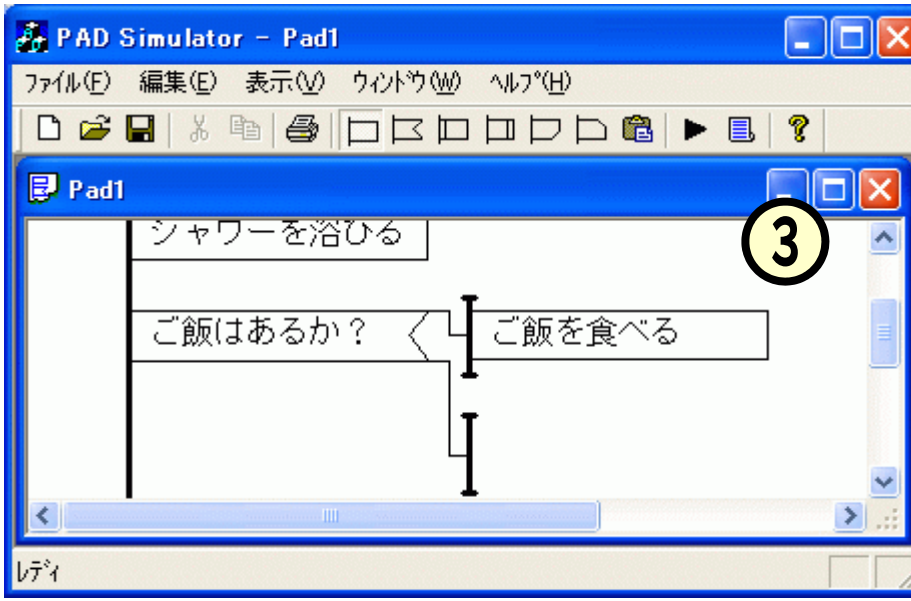


(3．2) 要素の挿入：分岐

■スライド (3.1) と同様な方法で、分岐 ( ) を選択すると、①のように、分岐の要素が追加されます。

■分岐の要素の右側に出来た縦棒 (②) に対しても、スライド (3.1) と同様な方法で、要素が追加できます (③)。

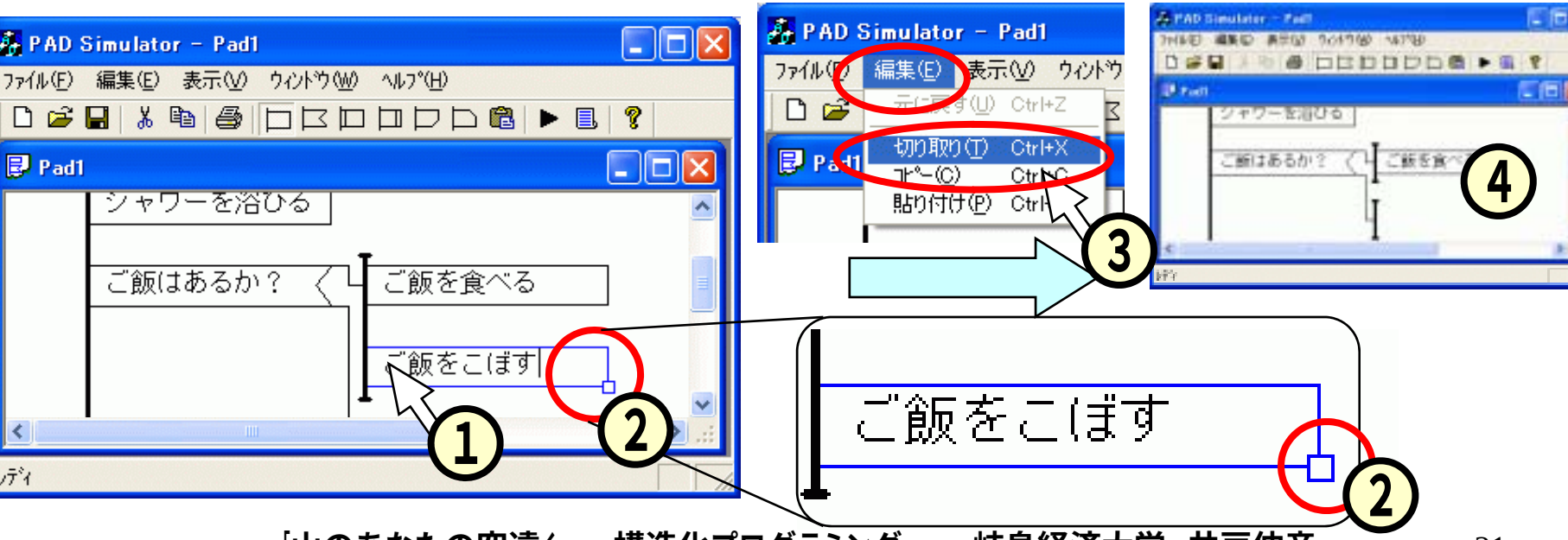




(3．3) 要素の選択、削除

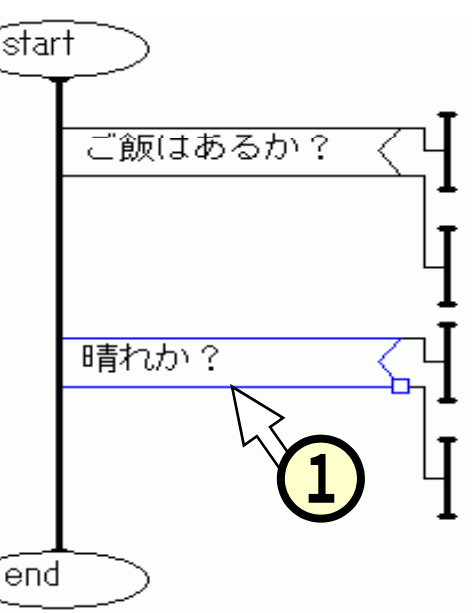
■選択　：　要素をクリック (①) して、選択します。選択すると、青い輪郭となり、右下に小さな四角 (②) が表示されます。

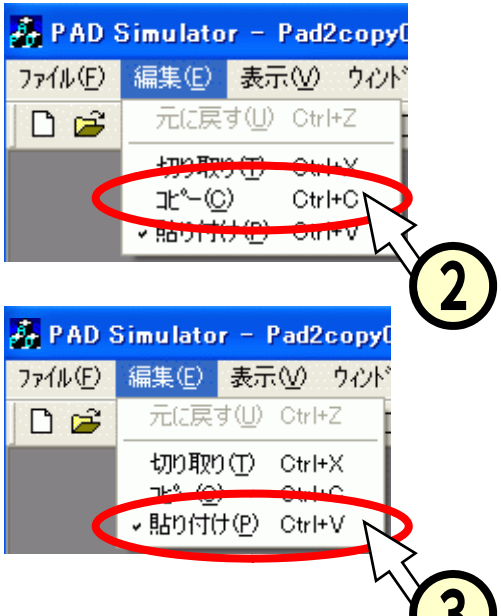
■削除　：　選択した状態で、[編集]-[切り取り]を選択 (③) すると、要素が削除されます (④)。Ctrl-x (Ctrl キーを押しながら、xキーを押す)でも、同じことです。

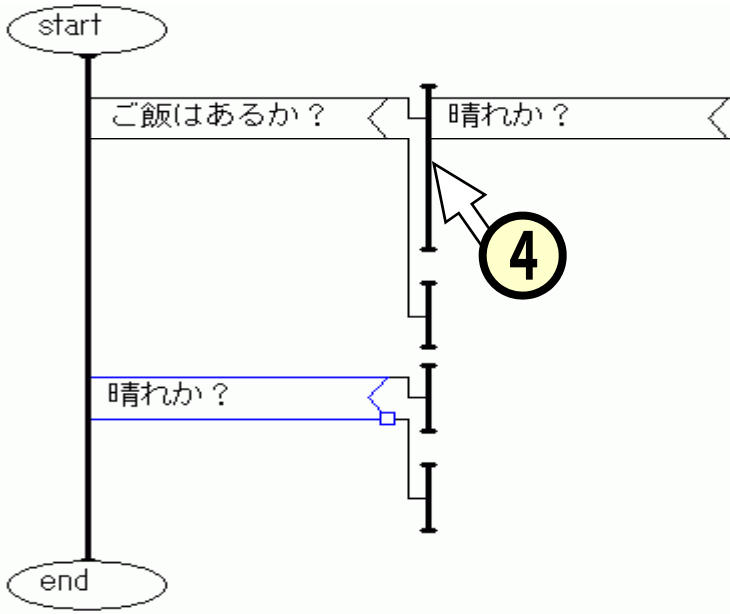


(3．4) コピー＆ペースト

■下図にて左から右のようにカット＆ペーストを行います。
■「晴れか？」の要素を選択 (①) します (青い輪郭)。
■[編集]-[コピー]をクリック (②) します (“Ctrl＋x”でもOK)。
■続けて、[編集]-[貼り付け]をクリック (③) します (“Ctrl＋v”でもOK)。
■貼り付けたい場所をクリック (④) します。
■カット＆ペーストも、同様の要領で行えます。







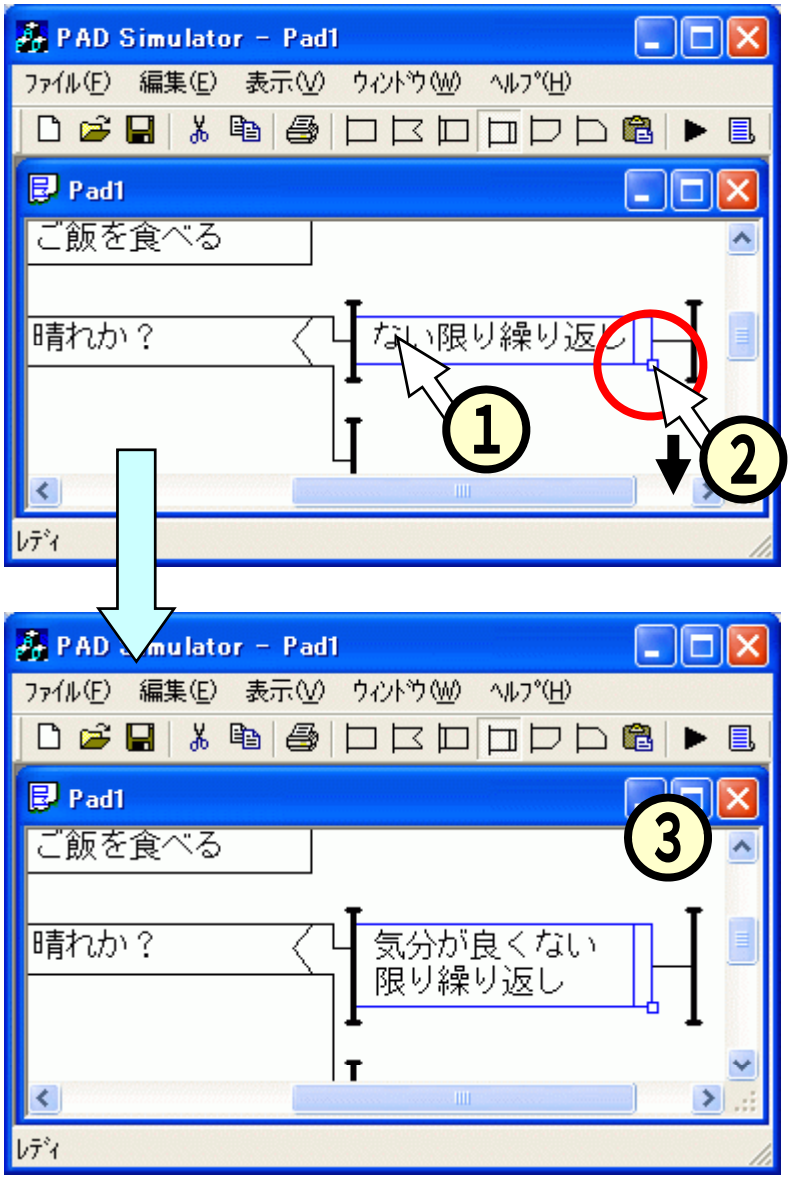
(3．5) 要素のサイズ変更、その他

■サイズ変更

- 要素を選択 (①) します。
- 要素の右下の小さい四角をドラッグ (②) すると、要素が広がります (③)。

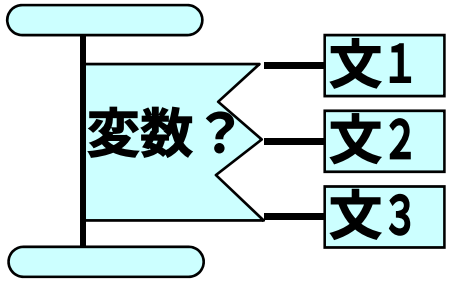
■その他

- ファイルの保存、読み込み等は、[ファイル]メニューより行うことが出来ます。
- 選択においては、上側がYesということになります。



(3．6) 注意：switch文

- C言語やJavaでの“switch文”を使うことは、プログラムの了解性を高める上で非常に役に立ちます。
- 残念ながら、PadSimulatorではこの”switch文”をサポートしていません。
- PADでは右図のように記します。
- 本講座では、“switch文”に相当するような処理を避けていますが、「“switch文” はなるべく利用したほうが良い」ということは、覚えておいてください。




```
1. switch(変数){
2.    case 値1:
3.      文1;
4.      break;
5.    case 値2:
6.      文2;
7.      break;
8.     case 値3:
9.      文3;
10.     break;
11.}
```

(3．7．1) 演習 (課題3ー1，3ー2)

■PADを作成してみましょう。

■課題3ー1：スライド (2.2) のPADを作成してください。

■課題3ー2：次に記したBさんの風邪の対処法について、PADを作成してください (故意に少し分かりにくく書いています)。

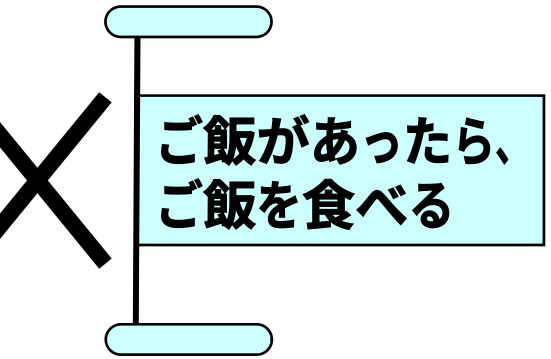
- 目を覚まして元気があると、遊びに出かける。
- 目を覚まして元気でないときは、食事をする。
- 食事をすませたら、体温を測る。
- 体温を測って熱があった場合、風邪薬を飲んで眠る。熱が無かった場合は、卵酒を飲んで眠る。

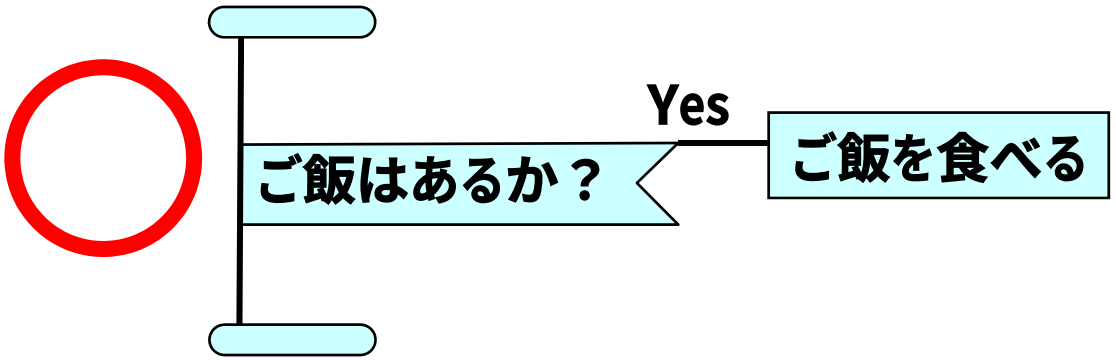
(3．7．2) ヒント (課題3ー2)

■故意に分かりにくく書いた訳ですが、これは「やりたいことの本質を見極めて、手順を整理する」ことの練習のつもりです。

- 繰り返されるのは、[目が覚める]から[寝る]までの行動であることのようです。
- 遊びに出かけると、風邪の対処法としては終了するようです。

■処理の中に「xxxだったら、yyyyする」と書いてはいけません。PADを使って表現してください。





(3．8) 演習 (課題3－3)

■次に記した、法案成立に向けたアプローチについて、PADを作成してください。

- 法案を作成して、委員会に諮る。
- 委員会の採決にて法案が否決された場合、法案を修正して再度委員会に諮る。
- 委員会の採決にて法案が可決された場合、本会議に諮る。
- 本会議の採決にて法案が否決された場合は、本会議を解散して総選挙を行う。
- 本会議の採決にて法案が可決された場合は、法案成立を宣言する。

(3．9) 演習 (課題3－4)

■テスト答案の採点と平均点算出の手順について、PADを作成してください (故意に少し分かりにくく書いています)。

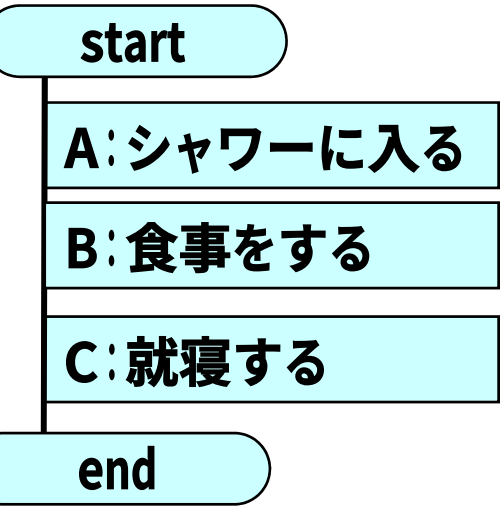
- 答案の束があるとします。
- これを一枚ずつ採点していきます。
- 当たり前ですが、平均点は次のようになります。
  - (平均点) ＝ (合計点) ／ (答案の枚数)
- (合計点) と (答案の枚数) とをどのように求めるかが、判るような手順としてください。

(4．1) 記述の粗さ／細かさ

■下に描かれた2つのPADを見比べて見てください。

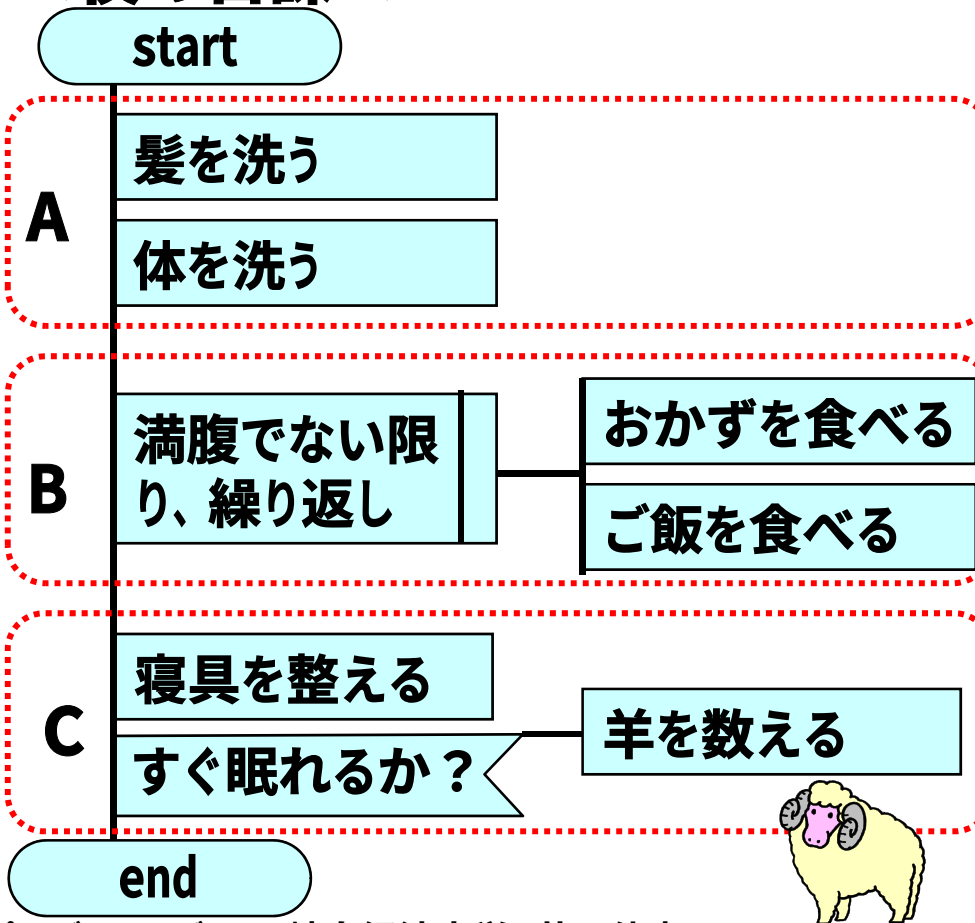
- 1と2とは、いずれも夜の日課について記しており、矛盾している訳ではありません。
- 1の方が粗い記述、2の方は細かい記述になっています。

＜夜の日課1＞



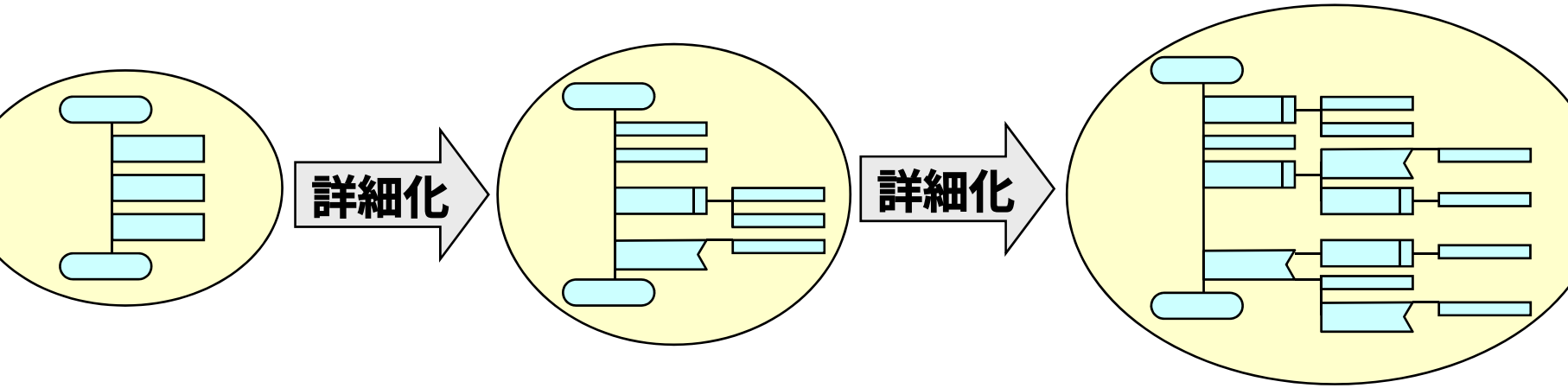
粗い ← 記述 → 細かい

＜夜の日課2＞



(4．2) 概要から詳細へ

■プログラムの設計を行う際は、“粗い”レベルから、“細かい”レベルへと考えて行きます。



■このようなアプローチを“段階的詳細化”と言います。

■粗いレベルのPADを、“概略PAD”、
細かいレベルのPADを、“詳細PAD”とも呼びます。
(どのレベルならば“詳細”というのかについては、
はっきりした定義はありません)

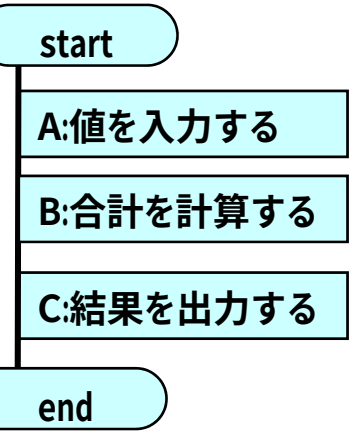
(4．3．1) PADとプログラムとの対応

■ここでやっと、プログラムについて考えることにします。

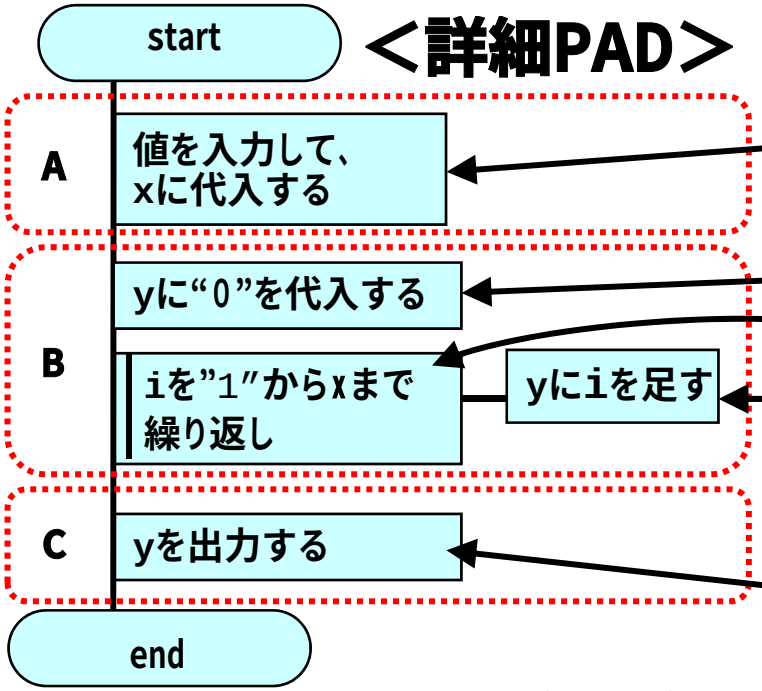
■プログラムに対する最も詳細化されたPADは、ソースコードに1：1に対応します。

■下図に、“1から入力した数までの合計値を出力する”プログラムの例を示します (C言語を用いています)。

<概要PAD>



<詳細PAD>



<ソースコード>

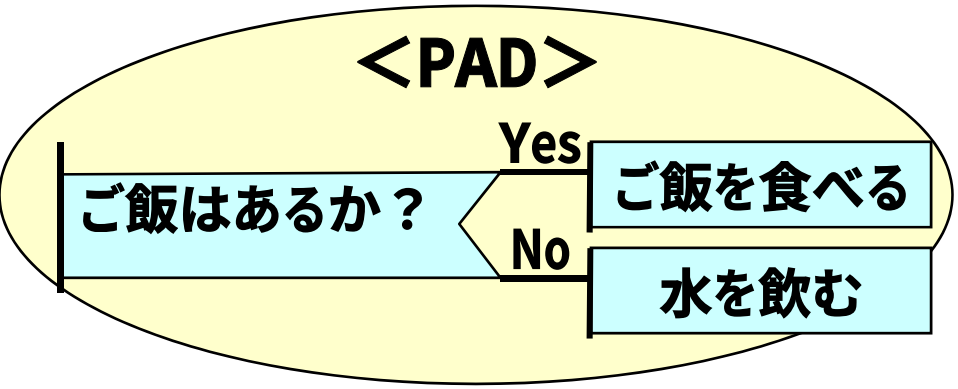
```
scanf(“%d”,x);

y=0;
for(i=0;i<=x;i++){
  y+=i;
}

printf(“%d¥n”,y);
```

(4．3．2) if文、while文

■容易に想像がつくとおり、PADとプログラムとの対応は次のようになっています。

■分岐⇔if文

<PAD>

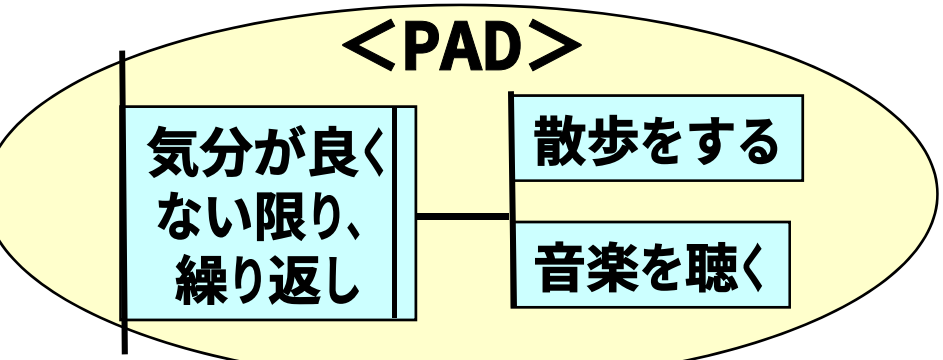


<ソースコード>

```
if(ご飯はあるか？){
   ご飯を食べる;
}else{
   水を飲む;
}
```

■繰り返し⇔while文 (もしくはfor文)

<PAD>

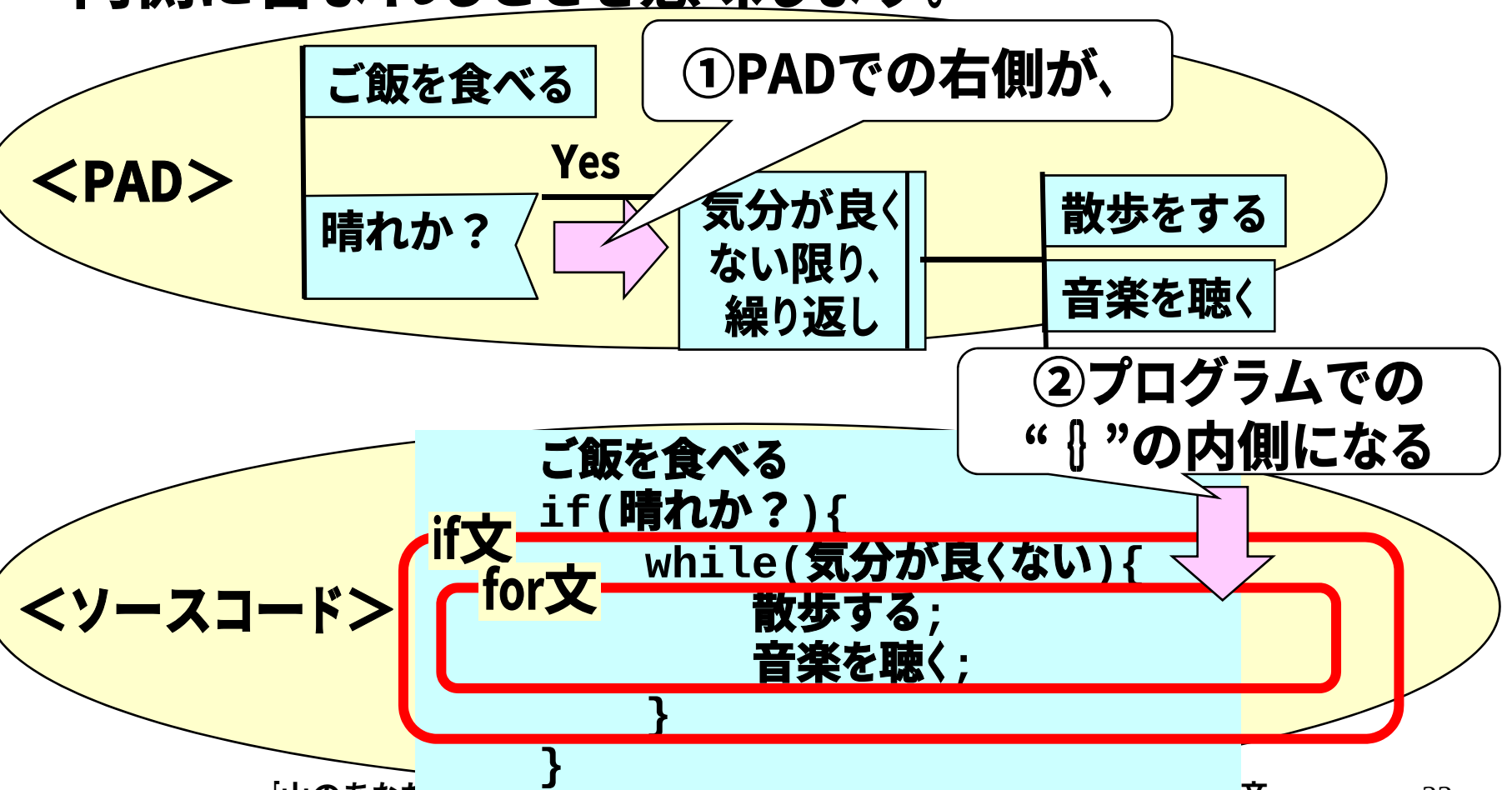


<ソースコード>

```
while(気分が良くない){
   散歩をする;
   音楽を聴く;
}
```

(4．3．3) ネスト(入れ子)

■PADで要素が右方向に連なる関係は、プログラムで言えばその要素に相当する処理が、“ {} ”(かっこ)の内側に含まれることを意味します。



(4．3．4) 段下げ

■プログラムを書く際、教師から「段下げを守って書くように」とひつこく言われたと思います。

■このような段下げは、PADで記した場合のような構造がプログラムで見えるようにするためのものです。

クラス
メソッド
for文
if文

```
public class Hello{
    public static void main(String args[]){
        int sum = 0;
        for(int i=0;i<10;i++){
            if((i % 2) ==0){
                sum += i;
            }
        }
        System.out.println("sum="+sum);
    }
}
```

(4．4) ツールの機能

■PadSimulatorには、詳細PAD(=言語レベルで描かれたPAD)を、C言語のサブセット (=簡略化された言語) のソースコードプログラムを生成する機能があります。詳細PAD自体を実行させる機能もあります。本講座では扱いませんが、各自トライしてみてください (取扱説明書を良く読んでください)。

＜言語レベルのPAD＞

生成

＜ソースコード＞

```
scanf(“%d”,x);

y=0;
for(i=0;i<=x;i++){
  y+=i;
}

printf(“%d¥n”,y);
```

■PadSimulatorのようなツールは、他にも存在します。ソースコードからPADを生成するツールもあります。ソースコードを直接見る代わりに、生成されたPADを見てプログラムを理解するためのものです。

＜ソースコード＞

```
scanf(“%d”,x);

y=0;
for(i=0;i<=x;i++){
  y+=i;
}

printf(“%d¥n”,y);
```

生成

＜言語レベルのPAD＞

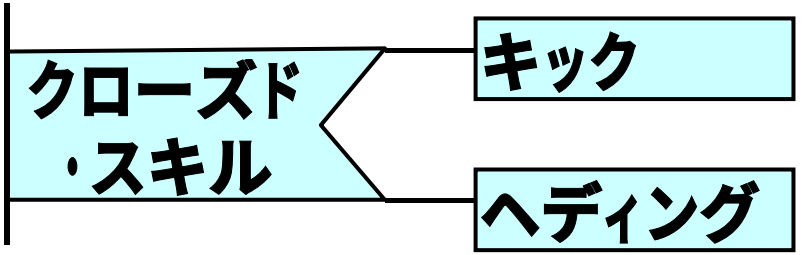
(4．5) 演習 (課題4ー1)

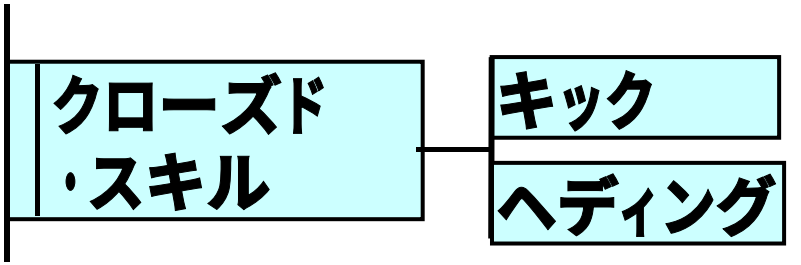
■課題4ー1：次の仕様を満たすプログラムの、概要PAD、詳細PADを順に作成してください。

- サッカー部の練習は、ウォーミングアップ、スキル練習、ミニゲーム、クールダウンの順で行われる。
- ウォーミングアップは、屈伸、柔軟を交互に3セット行う。
- スキル練習では、クローズドスキル、
  オープンスキルを交互に4セット行う。
- 偶数日には、クローズドスキルとしてキック、ヘディングを行い、オープンスキルとしてボールキープ、壁パスを行う。
- 奇数日には、クローズドスキルとしてドリブル、シュート、オープンスキルとして、スクリーンプレイ、オーバーラップを行う。
- 雨が降ったら、ミニゲームは中止する。
- ミニゲームをした時のみ、クールダウンを行う。

(4．5．1) ヒント (課題4ー1)

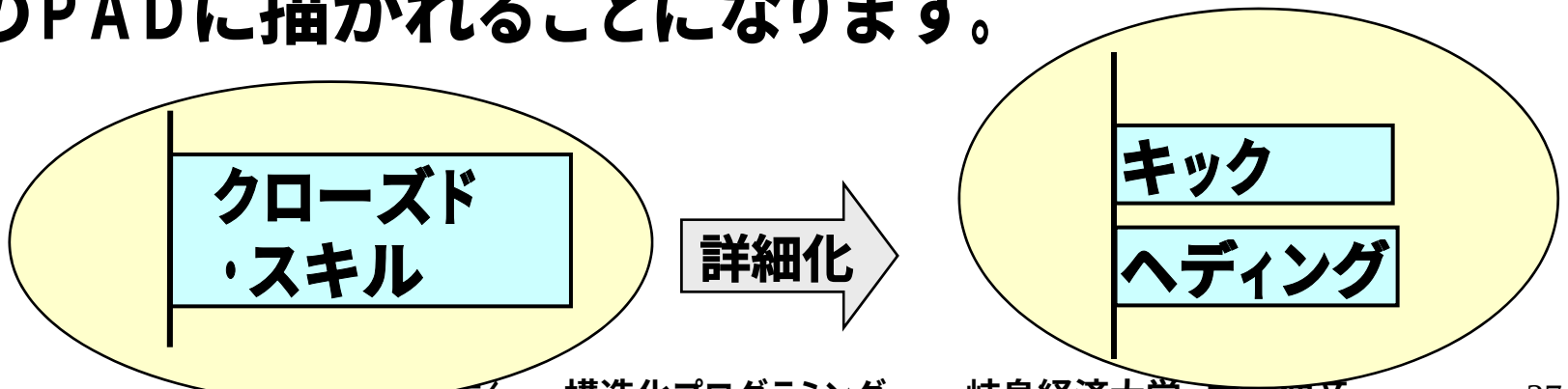
■次のような書き方は間違いです。





■スライド(2.5.3)「反復」、(2.5.5)「分岐」に記したような内容になっていないことは、明らかです。

■「クローズド・スキル」と、「キック、ヘディング」との関係は、詳細化になっています。ですから、次のように、別のＰＡＤに描かれることになります。



(4．6) 演習 (課題4ー2)

■課題4-2：次の仕様を満たすプログラムを、概要PAD、詳細PADの順に作成してください。

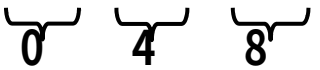
- 入力は、短い英数字文字列と、長い英数字文字列、整数nの3つ。
- 長い文字列の中に、次の文字列が現れたら、その現れた場所 (=何文字目か？) を出力する。
  - ◆短い文字列とn文字以下しか違わない文字列
- 例1：% kadai42 abc abdeatcssbc 1

  0　4　8
- 例2：% kadai42 bbca bbcbca 2

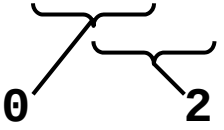
  0　2

(4．6．1) ヒント (課題4ー2)

- 例1：% kadai42 abc abdeatcssbc 1



  - 長い文字列の0から3文字は“abd” ⇒ “abc”と1つ違う ⇒0を出力
  - 長い文字列の1から3文字は“bde” ⇒ “abc”と3つ違う
  - 長い文字列の2から3文字は“dea” ⇒ “abc”と3つ違う
  - 長い文字列の3から3文字は“eat” ⇒ “abc”と3つ違う
  - 長い文字列の4から3文字は“atc” ⇒ “abc”と1つ違う ⇒4を出力
  - 長い文字列の5から3文字は“tcs” ⇒ “abc”と3つ違う
  - 長い文字列の6から3文字は“css” ⇒ “abc”と3つ違う
  - 長い文字列の7から3文字は“ssb” ⇒ “abc”と3つ違う
  - 長い文字列の8から3文字は“sbc” ⇒ “abc”と1つ違う ⇒8を出力
- 例2：% kadai42 bbca bbcbca 2



  - 長い文字列の0から4文字は“bbcb” ⇒ “bbca”と2つ違う ⇒0を出力
  - 長い文字列の1から4文字は“bcbc” ⇒ “bbca”と3つ違う
  - 長い文字列の2から4文字は“cbca” ⇒ “bbca”と1つ違う ⇒2を出力

(4．7) 演習 (課題4ー3)

■課題4-1の練習の様子を標準出力に打ち出すJavaプログラムを作成してください。

- 右図のような出力を行うプログラムです。
- 言語は知っているもの何でもOKです。
- 「偶数日or奇数日」、「晴れているか？」は、変数として与えてください。

■出力の言葉や形が違っていても、構いません。

```
c:¥javawork>java Kadai4
ウォーミングアップ
  屈伸1回目
  柔軟1回目
        :
  屈伸3回目
  柔軟3回目
スキル練習
  クローズドスキル1回目
      キック
      ヘディング
  オープンスキル1回目
      ボールキープ
          :
  オープンスキル4回目
      ボールキープ
      壁パス
ミニゲーム
クールダウン
```

(4．7．1) ヒント (課題4－3)　－その1－

■Javaの場合、右下のような形になると思います。ファイル名は、”Kadai43.java”となります。

```
import java.io.*;
public class Kadai43{
  public static void main(String args[]){
    // 偶数日か？
    boolean even = true;
    // 晴れか？
    boolean noRain = true;
    // ここに出力を行うプログラムを書く
  }
}
```

■「Javaスライド」の次の部分を参照してください。

- (1.3) プログラムの作成と実行　　(2) データの出力
- (8) 分岐、if文　　(9) 繰り返し

(4．8) 演習 (課題4－4)

■課題4－4：課題4－2のプログラムを作成してください。

- 言語はJavaを使いましょう。
- 入力は、標準入力より読み込んでください。
- 次のような形になると思います。ファイル名は、Kadai6.javaとなります。

```
import java.io.*;
public class Kadai6{
  public static void main(String args[]){
    // 入力
    String shortStr = args[0];
    String longStr = args[1];
    int n = Integer.parseInt(args[2]);
    // ここに処理を行うプログラムを書く
  }
}
```

shortStrに短い文字列を格納

longStrに長い文字列を格納

nに違って良い文字数nを格納

■「Javaスライド」の次の部分を参照してください。

- (12.7) プログラム引数

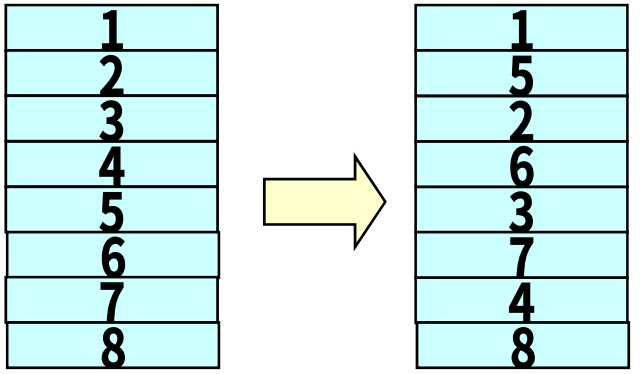
(4．9) 演習：シャッフル

■トランプのカードを切る動作を考えます。

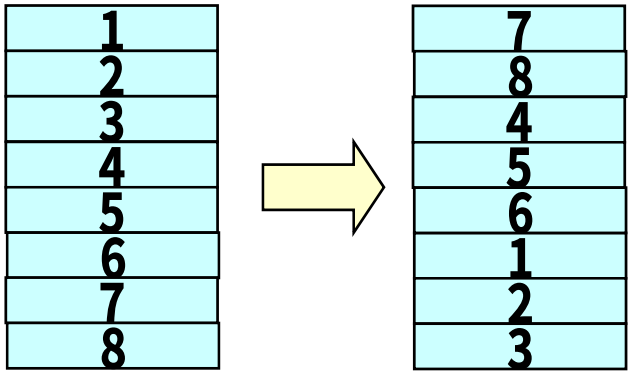
- リフル・シャッフル (Riffle Shuffle)
  - ◆カードを二組に分け、交互に重ね合わせていくきり方です。
- ヒンズー・シャッフル (Hindu Shuffle)
  - ◆花札などを切るときに使うきり方です。

■リフル・シャッフル、ヒンズー・シャッフルを指定しただけ行うプログラムを考えます。

＜リフル・シャッフル＞



＜ヒンズー・シャッフル＞



上記の例では、切り幅を3枚としている。

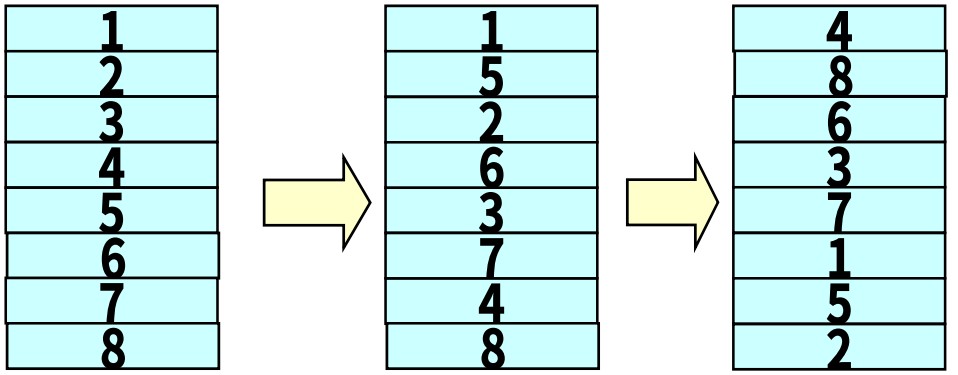
(4．9．1) 演習：プログラムの仕様

■入力は、①切り方を表す“r”と“h”とから成る文字列と、②カードの枚数を表す整数値の2つ。

■カードは、1からカードの枚数までの番号が付けられているとする。切り方を表す文字列に従って順にシャッフルした場合に、カードがどのような並びになるかを出力する。ヒンズー・シャッフルでの切り幅は、3に固定する。

- 例1：% kadai7 r 8
  1,5,2,6,3,7,4,8, by r
- 例2：% kadai7 h 8
  7,8,4,5,6,1,2,3, by h
- 例3：% kadai7 rh 8
  1,5,2,6,3,7,4,8 by r
  4,8,6,3,7,1,5,2 by h

＜例3＞



(4．9．2) 課題4－5、課題4－6

■ (課題4－5) カードの枚数は、6の倍数に限ることにします。

- Javaによるプログラム(Kadai45.java)を作成してください。

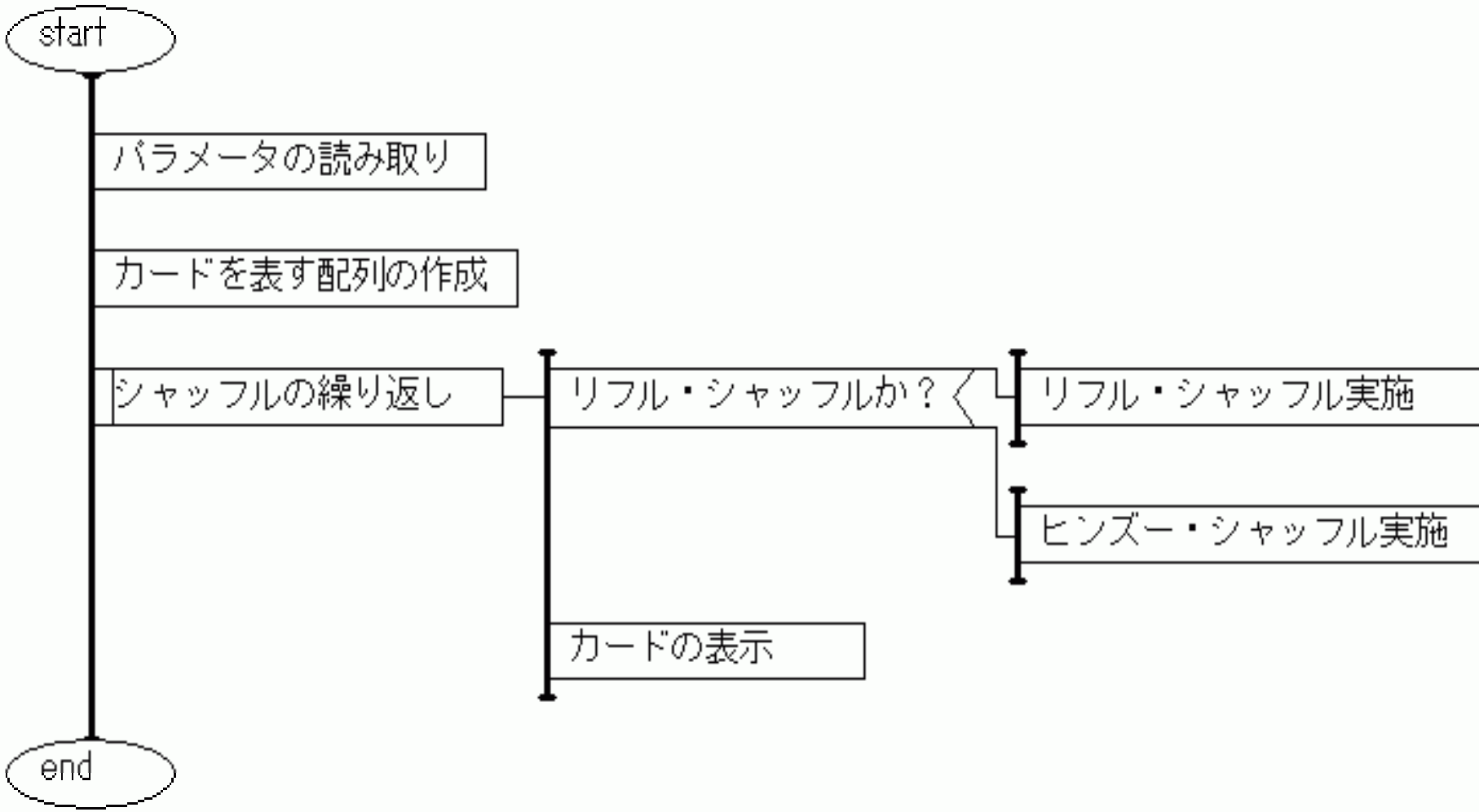
■ (課題4－6) カードの枚数を、任意とします。

- Javaによるプログラム(Kadai46.java)を作成してください。

※この課題は、ＰＡＤ作成の練習を意図したものです。このため、本スライドの解答例は手続き型に徹したものにしています。現実的には、オブジェクト指向の考えに基づく解答が現在では自然かつ容易なものとなります。

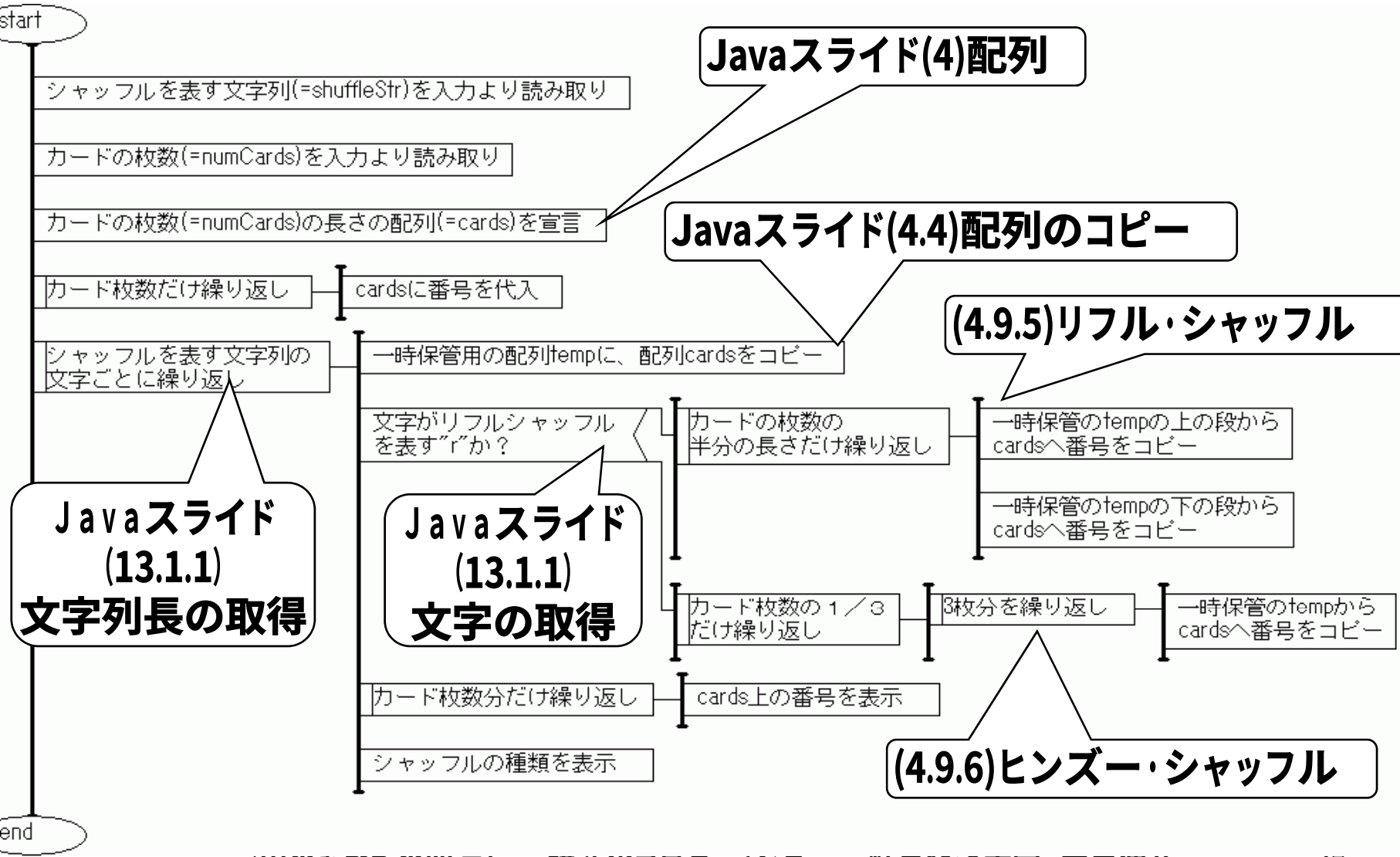
(4．9．3) 課題4－5：ヒント1

■概略ＰＡＤを示します。



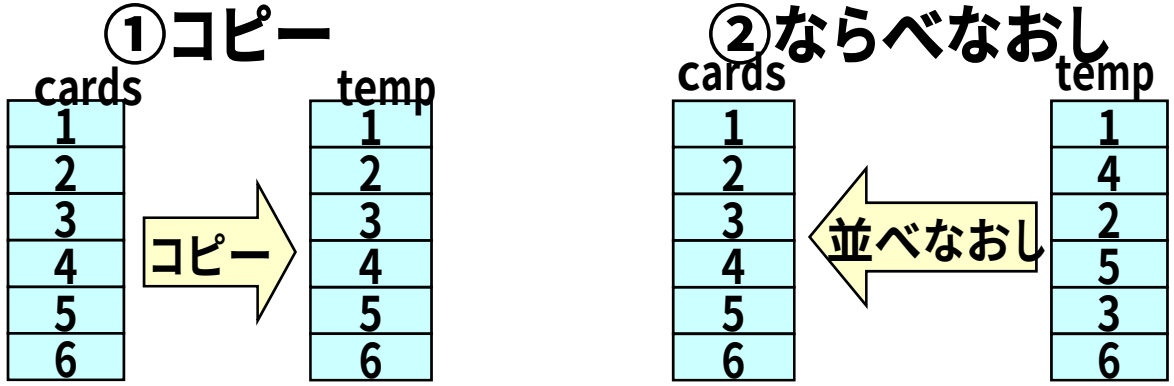
(4．9．4) 課題4－5：ヒント2

■詳細ＰＡＤ

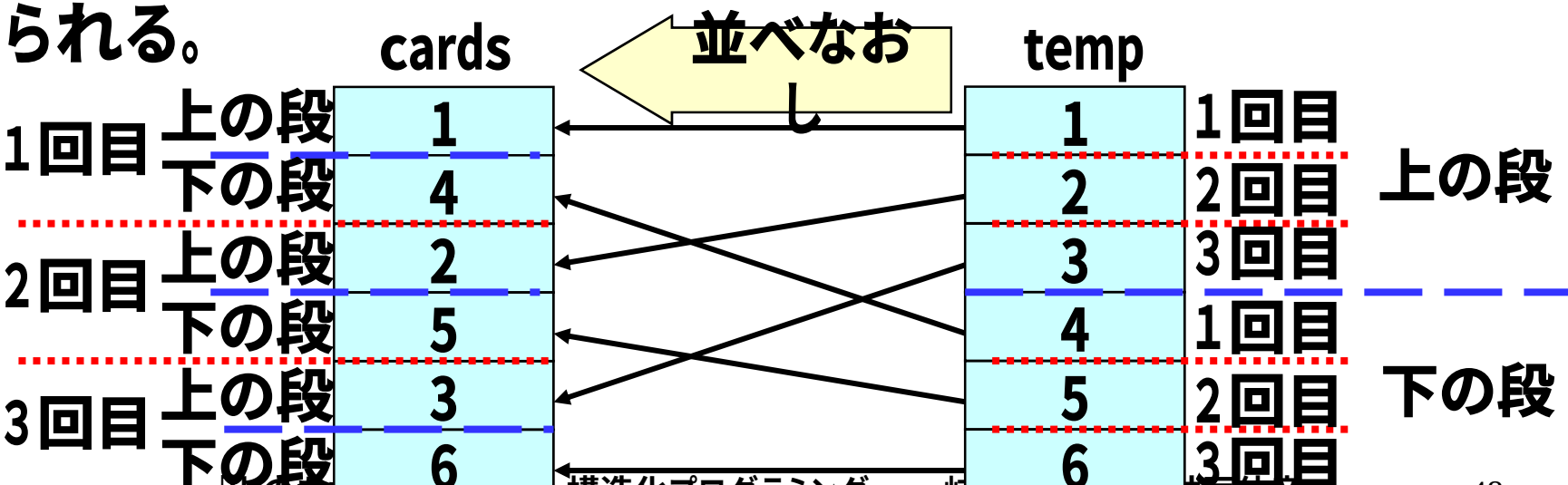


(4．9．5) 課題4－5：リフル・シャッフル

■①まず一時保管場所にコピーして、②そこから並べなおすことにする。



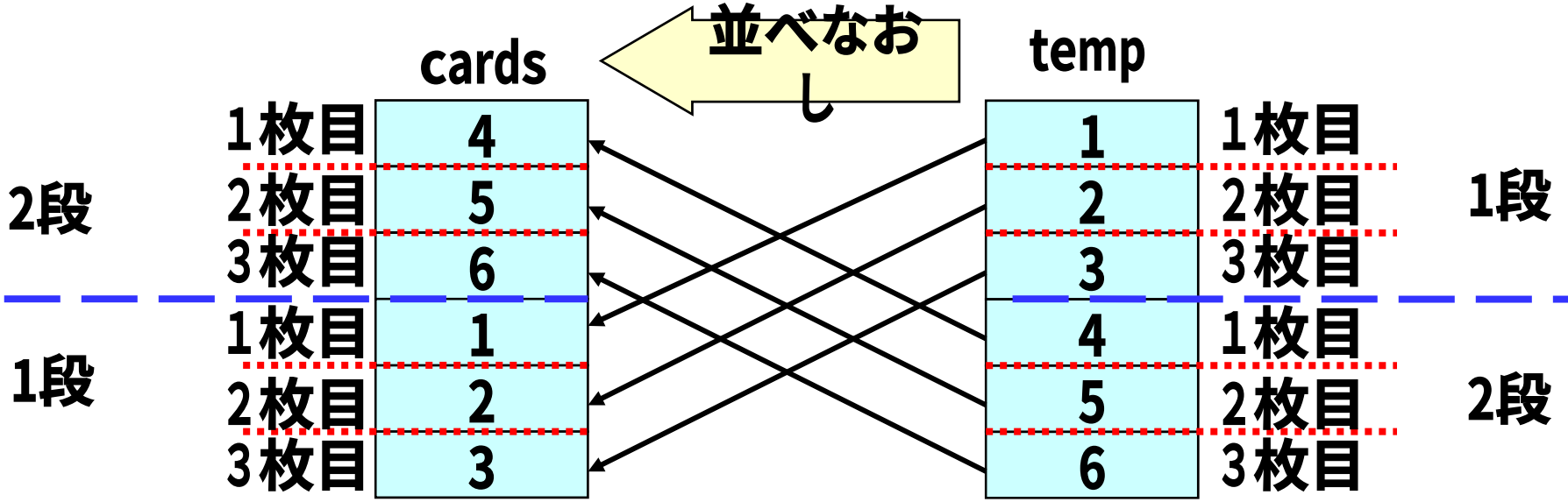
■枚数は6の倍数だから、2で割り切れる。よって、シャッフルする前のカードは、上の段と下の段とに分けられる。



(4．9．6) 課題4－5：ヒンズー・シャッフル

■枚数は6の倍数だから、3で割り切れる。よって、シャッフルする前のカードは、段に分けられる。

■各段について、3枚ずつ並べなおしていけば良い。



(4．10) 演習：オセロゲーム

■ここで考えるオセロゲームとは、次のようなゲームです (一般的なルールを簡略化しています)。

- 8行8列、64マスの盤を用います。
- ゲーム開始時、マス目には何も置かれていません。
- 黒が先手、白が後手となって、交互に石を打ちます。
- 縦・横・ななめ方向に相手色 (例えば黒) の石を自分色 (例えば白) ではさむと、挟まれた石を自分色 (例えば白) に返すことができます。
- 石を置く場所が無くなった時点でゲームは終了し、盤上に石の多い方が勝ちとなります。

■一般的なルールとの主な違いは、次の点です。

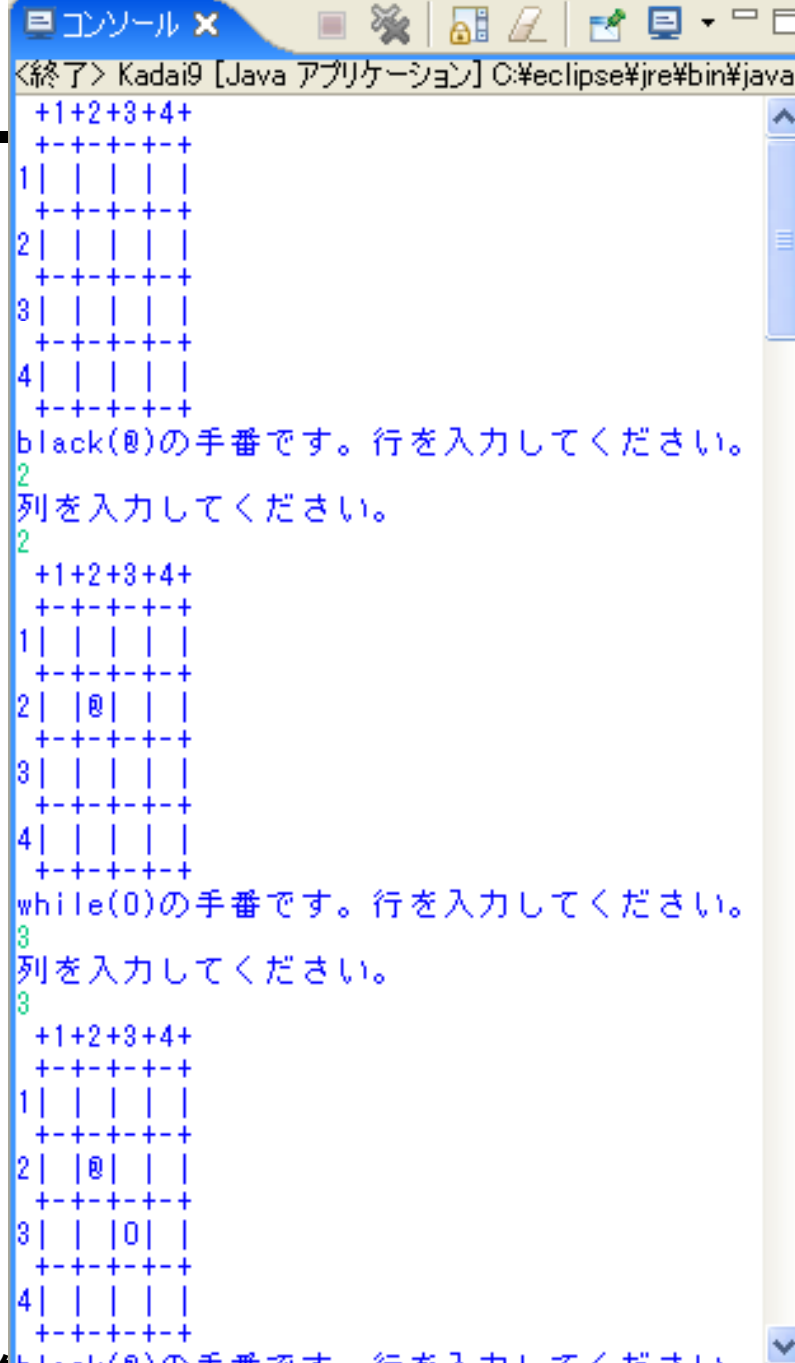
- (1) 最初に黒2つ白2つの石を中央に置きます。
- (2) 必ず返しが起こる手を打たねばならず、そうで無い場合は自分の番をパスします。
- (3) 返しが起こる手があるのにパスをすることは出来ません。

(4．10．1) プログラム

■右図のように、同じウインドウに黒白交互に石を置く位置を選ぶスタイルの、オセロゲームのプログラムを作成します (右図では、表示のために4行4列の盤としています)。

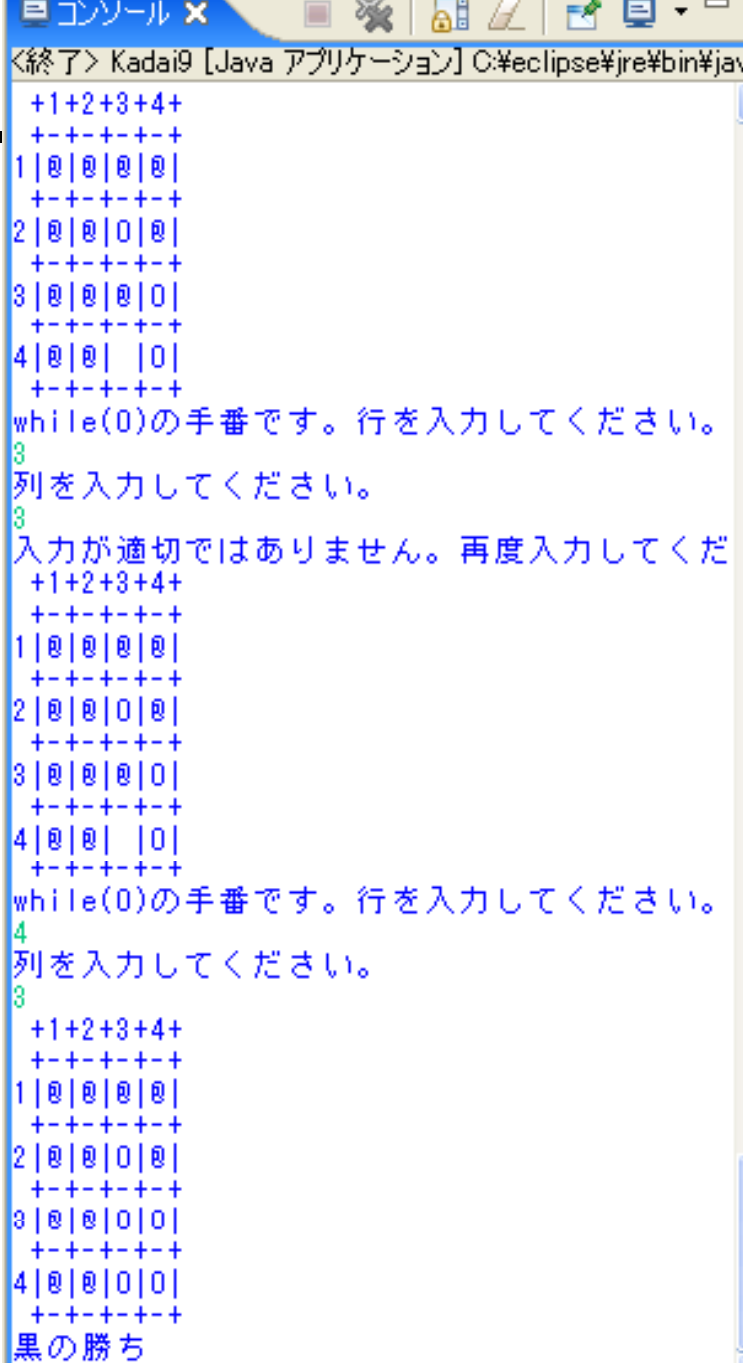
■黒・白の石は、次の文字で表すこととします。

- 黒：@　(アットマーク)
- 白：O　(大文字のオー)



(4．10．2)プログラム

- すでに石が打たれている位置を指定された場合は、再度入力するように促します。
- 数字以外の入力があった場合については、考慮する必要はありません。
- ゲーム終了時には、勝敗の判定を表示します。



(4.10.3) 課題

■ (課題4－7)

- 概略PADを作成してください。

(詳細ＰＡＤはかなり面倒ですので、この場合は省略してもＯＫです。)

■ (課題4－8)

- Javaにてプログラムを作成してください。

■ (課題4－9)

- 一般的なルールとしてプログラムを作成してください。

(4．10．4)ヒント

■プログラム作成例の一部分を示します。

```
      (略)
public class Kadai46{
    private static final int SIZE_BOARD=8;
    private static final int NONE=0;
    private static final int BLACK=1;
    private static final int WHITE=2;
    private static final String[] bAndW
    ={"none","black","while"};
    private static final String[] Syms={" ","@","O"};

  public static void main(String args[]){
    int[][] board=new int[SIZE_BOARD][SIZE_BOARD];
       (略)
  }
  private static void drawBoard(int[][] board){
       (略)
  }
}
```

盤の大きさ

石無しを0で、
黒を1で、白を2で、
それぞれ表す。

石の印に用いる文字

盤上の石を覚えておく配列：値は、NONE(=0),BLACK(=1),WHITE(=2)

盤上の石を覚えておく配列を引数として、
盤の表示を行うメソッド

(4.11) 演習 (課題4ー10)

■課題4-10：次の仕様を満たすプログラムを、概要PAD、詳細PAD、プログラムの順に作成してください。

- 入力は、短い英数字文字列と、長い英数字文字列の2つ。
- 長い文字列の文字を、短い文字列にもあるものだけ、順に出力する。
- 例1：% Kadai410 aiueo ktadifodnifde
  aioie
- 例2：% Kadai410 12345 sd98348fa36
  343

(4．12．1) 演習 (課題4ー11)

■課題4-11：次の仕様を満たすプログラムを、概要PAD、詳細PAD、プログラムの順に作成してください。

- 入力は、英数字文字列の1つだけ。
- 入力した文字列から、重複する文字を取り除いた文字列を出力する。
- 例1：% Kadai411 aabkdidfdaijiia

  abkdifj
- 例2：% Kadai411 1231234556

  123456

(4．12．2) continue文(課題4-11の補足)

■この問題では、次のようなcontinue文を用いることにします。

```
1 : LP_X:for(int i=0;i<5;i++){
2 :    if(i==3){
3 :      continue LP_X;
4 :    }
5 :    System.out.print(i);
6 : }
```

// “0124”と出力される。

2行目へ
スキップ
(i=4となる)

■PAD中では、「繰り返しLP__X」とか、「LP__Xの先頭へスキップする」とか、記してください。

(5．1) goto文

■プログラミングを学んだ際、おそらく“goto文”を教わらなかった人も多いと思います。

■goto文とは、これを実行すると、指定されたプログラム行に制御が移るような制御文です。

ラベルの位置へ飛んでいく

```
scanf(“%d”,x);
y=0;
i=0;
Label:i+=1;
   y++;
   if(i>10){
      goto Label;
   }
}
printf(“%d¥n”,y);
```

■昔のプログラムには、goto文を多用したものがあり、これがプログラムを分かりにくくする原因になっていました（元々アセンブラには、if文とgoto文しかありません．フローチャートもそうです）。

■ダイクストラ(E. W. Dijkstra)が「goto文を使わない」ことを提唱し、以来、プログラミングにおける悪者として避けられてきました。皆さんが教わっていないのは、このためです。

(5．2．1) 思考実験　ーその1ー

■右下図のプログラムについて、変数x,yが次のような値であった際の出力を考えてください。 <ソースコードA>

- (例) x=1,y=1
  - ◆[出力]
    - □foo1
    - □foo5
- x=1,y=-1
- x=-1,y=1
- x=-1,y=-1

■それほど難しい訳ではありませんね。

```
if(x>0){
  printf(“foo1”);
  if(y<=0){
    printf(“foo4”);
  }
  printf(“foo5”);
}else{
  if(y>0){
    printf(“foo2”);
  }else{
    printf(“foo3”);
    printf(“foo5”);
  }
}
```

“printf(“xx”)”は、xxを出力する命令です。

(5．2．2)思考実験　ーその2ー

■goto文を使った右下図のプログラムについて、同様に変数x,yが次のような値であった際の出力を考えてください。

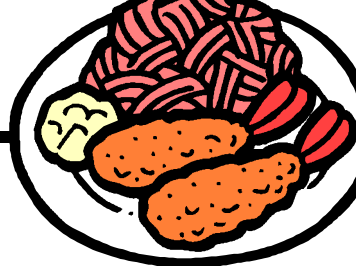
- (例) x=1,y=1
  - ◆[出力]
    - □foo1
    - □foo5
- x=1,y=-1
- x=-1,y=1
- x=-1,y=-1

■今度は容易ではありません。

＜ソースコードB＞

```
if(x>0){
  printf(“foo1”)
  goto Lavel1;
}
if(y>0){
  printf(“foo2”);
  goto Lavel2;
}
print(“foo3”);
Lavel1:
If(x>0 && y<=0){
  printf(“foo4”);
}
printf(“foo5”);
Lavel2:
```

(5．2．3) スパゲッティ



■goto文は何が悪いか？

- 実は、(5.2.1)プログラムAと(5.2.3)プログラムBとは同じ動きをします。しかしながら、Aの動きは明瞭であるのに対し、Bの動きを追うのには骨が折れます（わざとらしい例で恐縮ですが）。
- Bのように制御が飛ぶ流れが複雑に絡み合うプログラムを、“スパゲッティ・プログラム”などと言っていました。

＜ソースコードA＞

```
if(x>0){
  printf(“foo1”);
  if(y<=0){
    printf(“foo4”);
  }
  printf(“foo5”);
}else{
  if(y>0){
    printf(“foo2”);
  }else{
    printf(“foo3”);
    printf(“foo5”);
  }
}
```

＜ソースコードB＞

```
if(x>0){
  printf(“foo1”)
  goto Lavel1;
}
if(y>0){
  printf(“foo2”);
  goto Lavel2;
}
print(“foo3”);
Lavel1:
If(x>0 && y<=0){
  printf(“foo4”);
}
printf(“foo5”);
Lavel2:
```

(5．3．1) PADに描けるか？

■ソースコードA、BのPADを作成して、両者を比較してみます。<ソースコードA>

```
if(x>0){
  printf(“foo1”);
  if(y<=0){
    printf(“foo4”);
  }
  printf(“foo5”);
}else{
  if(y>0){
    printf(“foo2”
  }else{
    printf(“foo
    printf(
  }
}
```

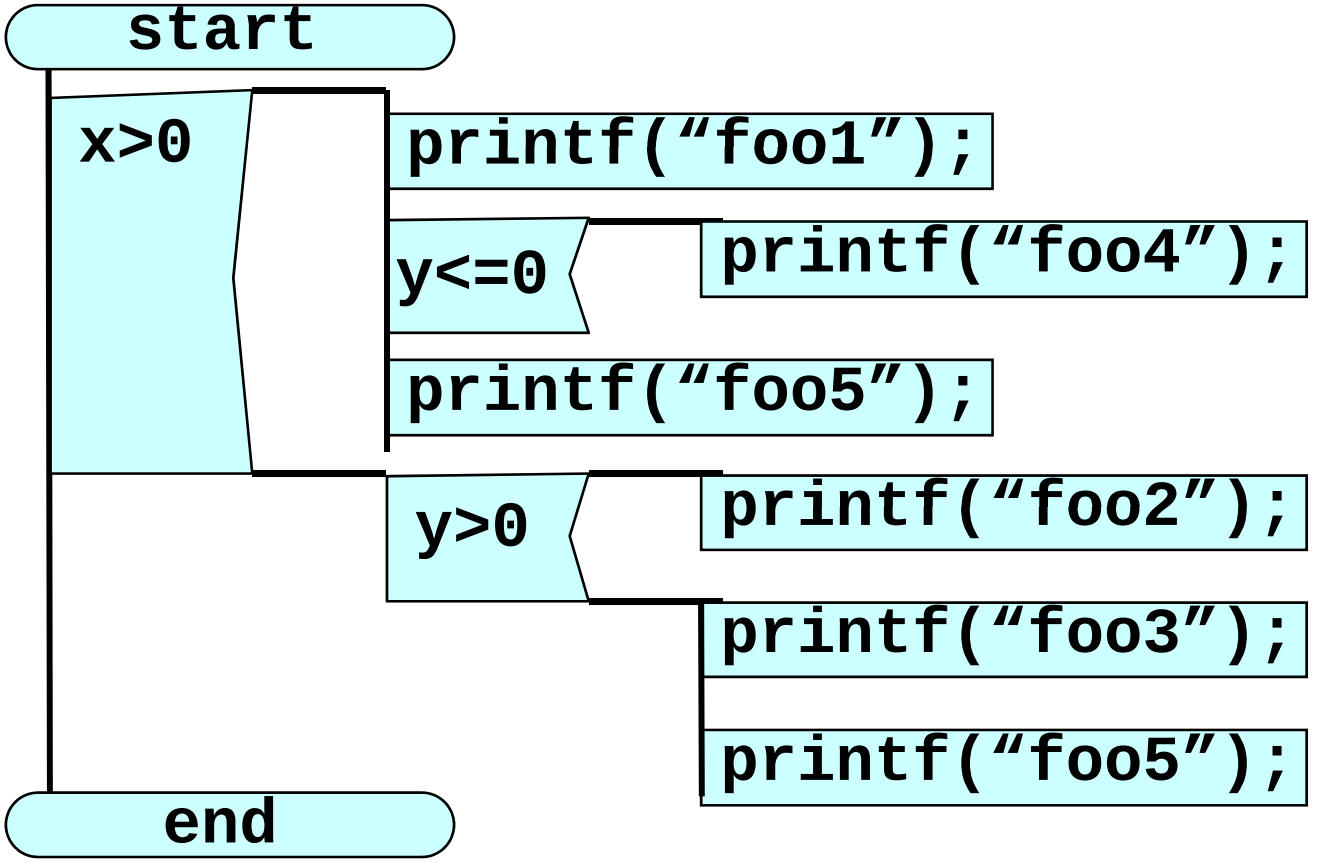
<ソースコードB>

```
if(x>0){
  printf(“foo1”)
  goto Lavel1;
}
if(y>0){
  printf(“foo2”);
  goto Lavel2;
}
print(“foo
Lavel1:
If(x>0 &&
  pr
}
printf(“fo
Lavel2:
```

■まず、ソースコードAについて作成してみてください。

(5．3．2)ソースコードAのPAD

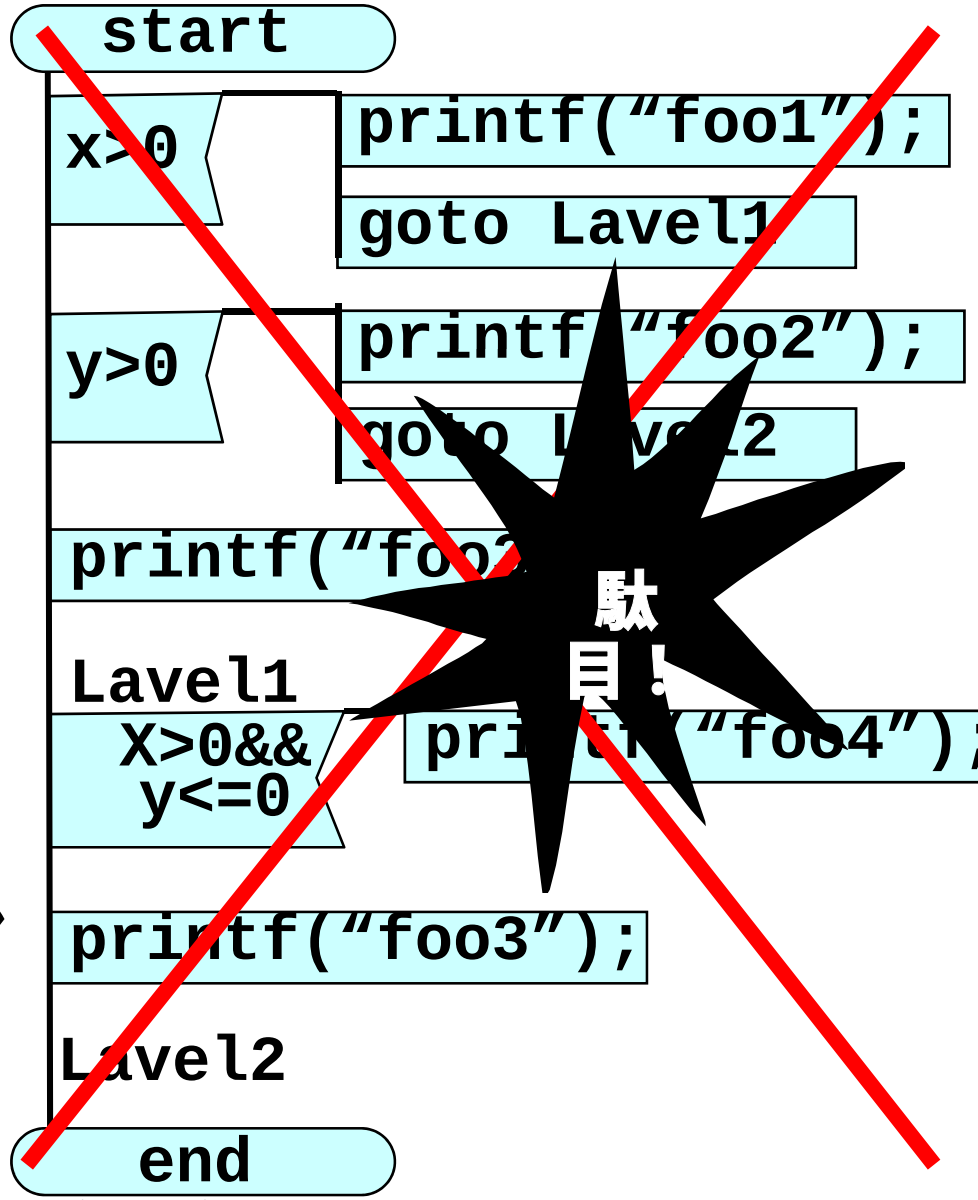
■ソースコードAについては、難なくPADが作成できることと思います。



■それではソースコードBのPADはどうなるでしょう？

(5．3．3) ソースコードBのPAD

- ソースコードBは、うまくPADに描けないことと思います。
- 無理やりラベルを書き込むことにすると、右図のようになります（こんな記法は無いのでまねしないでくださいね）。
- このようにしても、BのPADは、プログラムがどのように動くかについて、その形状からなにもヒントを得られません。



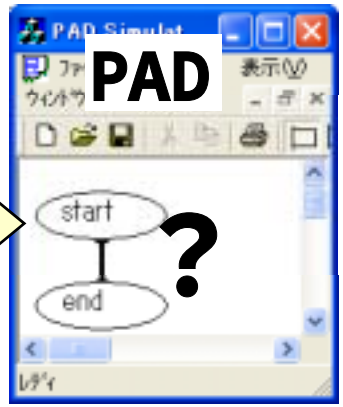
(5．3．4) goto文とＰＡＤ

■ソースコードA、BのPADを作成した結果は、次のようになりました。

＜ソースコードA＞

```
if(x>0){
  printf(“foo1”);
  );
}
  );
    printf(“foo3”);
    printf(“foo5”);
  }
}
```

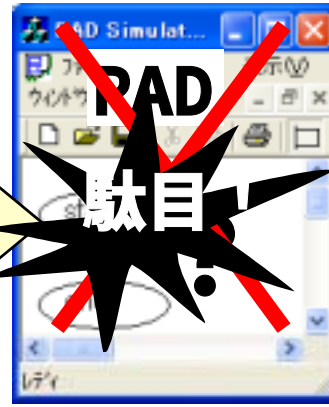
goto文を含まないプログラム



＜ソースコードB＞

```
if(x>0){
  printf(“foo1”)
  ;
}
If(x>0 && y<=0){
  printf(“foo4”);
}
printf(“foo5”);
Lavel2:
```

goto文を含む(悪い)プログラム



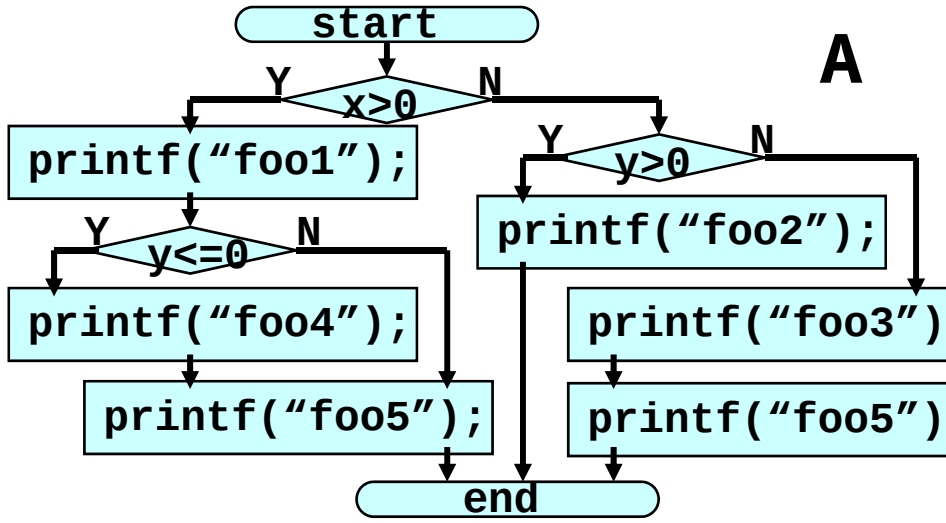
■つまり、goto文を含むBのようなプログラムは、PADが描けないのです。ＰＡＤはスパゲティ・プログラム(悪いプログラム)を防ぐ番人になっていました。

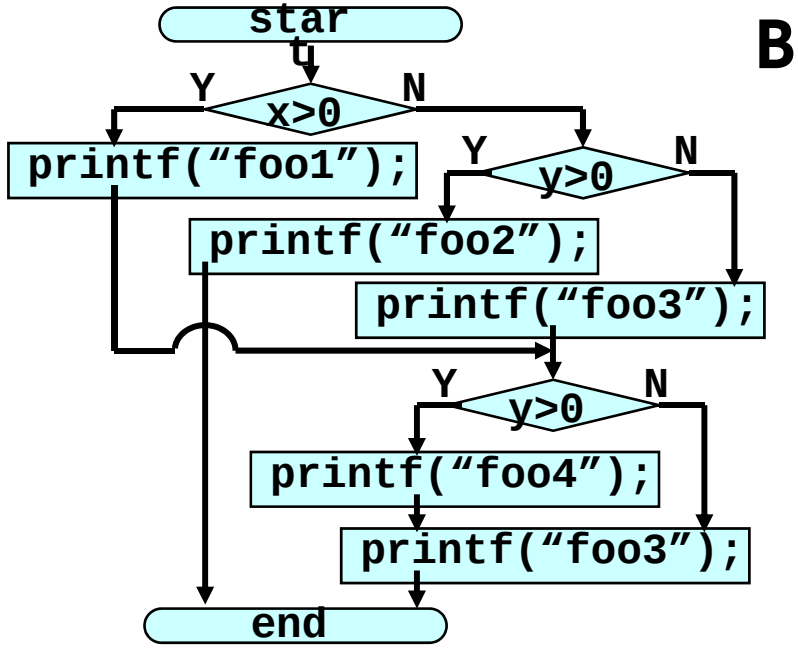
構造化されたプログラム(良い)

スパゲティ・プログラム(悪い)

(5．3．5) フローチャートに描けるか？

- ソースコードA、Bについて、フローチャートを描いて見ます。すると、A、Bとも普通に作成出来てしまいます。
- すなわち、フローチャートで設計を行うと、Bのような駄目なプログラムを設計してしまう恐れがあるのです。
- フローチャートよりPADを使うべきであること、PADがgoto文を使わないプログラムと密接に関係していることを理解して頂けましたか？





(5．4) goto文は不要か？

■ここまでは、goto文の話というよりも、PADの話でした。
ここからはgoto文そのものの話です。

■ところで、goto文は全く不要なのでしょうか？

■講師の考えを言うと、次のようになります。

- 「ラベルつきbreak文」をサポートするプログラミング言語(Java等)では、不要。
  - ◆Javaには、「ラベルつきbreak文」があり、実際、goto文は使えません。
- 「ラベルつきbreak文」をサポートしないプログラミング言語では、goto文が必要な場合もある。
  - ◆C、C++には、「ラベルつきbreak文」がありません。
  - ◆Perlには、「ラベルつきbreak文」はありますが、goto文も使えます。

(5．5) ラベル付きbreak文

■九九の結果を出力するプログラムを考えます。但し、指定された数が現れたら、そこで出力をやめることとします。

■2つのループを抜けるためには、C言語のようにラベル付きbreak文が無い言語では、＜A＞のようにgoto文を使うしかありません(goto文の使用を避ける方法もありますが、あまり勧められる方法ではありません)。

■JavaやPerlなどの言語では、Bのように「ラベル付きbreak文」を使えばよく、goto文を使う必要はありません。

■「Javaスライド (9.6) および(10.8)」してください。

```
/*指定値をxに読み込む*/
Scanf(“%d”,x);
for(i=1;i<10;i++){
  for(j=1;j<10;j++){
    y = i*j;
    printf(“%d “,y);
    if(y==x){
      goto Lavel;
    }
  }
}
Lavel:
```

＜A＞

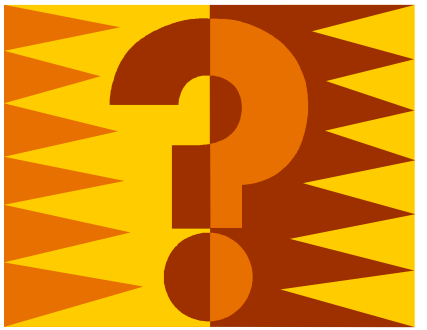
```
/*指定値をxに読み込む*/
Scanf(“%d”,x);
Lavel:
for(i=1;i<10;i++){
  for(j=1;j<10;j++){
    y = i*j;
    printf(“%d “,y);
    if(y==x){
      break Lavel;
    }
  }
}
```

＜B＞

(5．6．1) 演習

■次のように動作するプログラムを考えます。

```
% 入力:3ab
出力:ab ab ab
% 入力:ab 2bc cd
出力:ab bc bc cd
% 入力: 234 ab 1c
出力:34 34 ab c
% 入力:abcdefg 2hij k
出力:abcdefg hij hij k
% 入力:34b ido
出力:4b 4b 4b ido
% 入力:2icc 4bc
出力:icc icc bc bc bc bc
```



(5．6．2) 演習 (つづき)

■フローチャートを見てください。

start
文字列を入力
文字列の読出し位置を先頭に設定
読出し位置が文字列内で、その位置の文字が空白か？
No
Yes
読出し位置を進める
読出し位置が文字列内か？
No
Yes
単語を空にする
end
読出し位置が文字列内で、その位置の文字が空白でないか？
No
Yes
読出し位置の文字を単語に追加
読出し位置を進める
読出し位置が文字列内で、その位置の文字が空白か？
No
Yes
読出し位置を進める
単語の長さが0か？
Yes
No
単語の先頭文字は数字か？
No
Yes
単語と空白とを出力
カウンタを数字の値に設定
先頭文字を除いた単語と空白とを出力
カウンタから1をマイナス
カウンタはゼロか？
No
Yes

(5．7) 演習 (課題5－1，2，3)

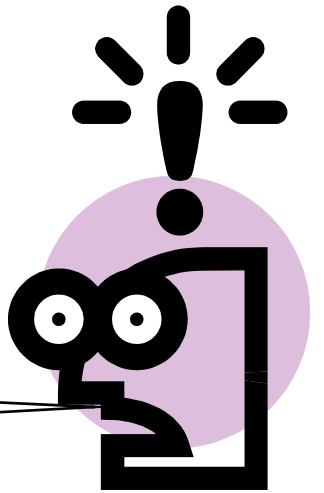
■課題5－1：前スライド (5.6.1) に描かれたプログラムは、何を入力して、何を出力しているかを説明してください。

- 文字列“abc 3cde 2c”を入力した場合の出力は何ですか？

■課題5－2：前スライド (5.6.2) のＰＡＤを作成してください。前スライド (5.6.2) のフローチャートと比較してみてください。

■課題5－3：課題5－1のプログラムを、作成してください。

- Ｊａｖａで作成します。
- 入力は、標準入力より読み込んでください。



(6) 関数 (サブルーチン、モジュール)

■関数 (サブルーチン、モジュール) には、3つの役割があります。

- [共通化] 共通の処理を、一箇所にまとめる。
- [規模の分割] プログラムを適当な大きさに分割する。
- [意味的な分割] プログラムを理解しやすくするため、意味のある単位に分割する。

■このうち、[意味的な分割]については、複数の考え方があります。“オブジェクト指向”というものも、そのような考え方のひとつと捉えることも出来ます (オブジェクト指向については、本講座では取り上げません) 。

■本講座では、階層的な考え方について見ていきます。

■なお、本 (6) 項は、“構造化プログラミング”と直接関係はありません。

(6．1)料理

■料理を作るプログラムを考えます。

■朝食、昼食、夕食の3食を作ります。

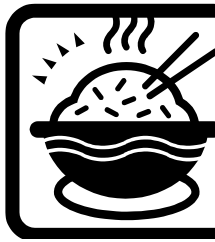
■献立は右表のとおりです。

朝食	味噌汁、ご飯、卵焼き
昼食	味噌汁、サラダ
夕食	シチュー、ご飯、サラダ

■味噌汁は、朝食･昼食とも献立にあるので、共通化出来るものとします。

```
make_breakfast(){
/*朝食を作る処理*/
 make_soup(xxx);/*味噌汁を作る*/
 make_rice(yyy);/*ご飯を作る*/
 make_egg(zzz);/*卵焼きを作る*/
}
```

```
make_soup(){
/*味噌汁を作る処理*/
      (略)
}
```

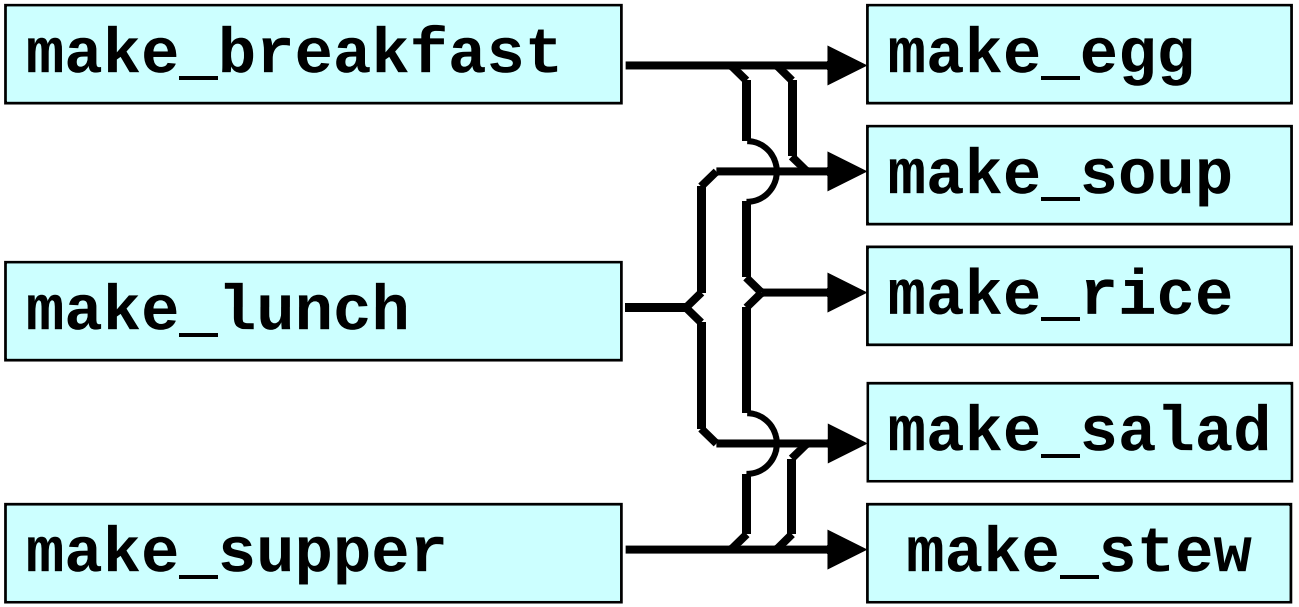


■このように、関数“make_breakfast”が、関数“make_soup”を呼んでいるとき、この関係を次のように記すこととします。

Make_breakfast → make_soup

(6．2) 共通化

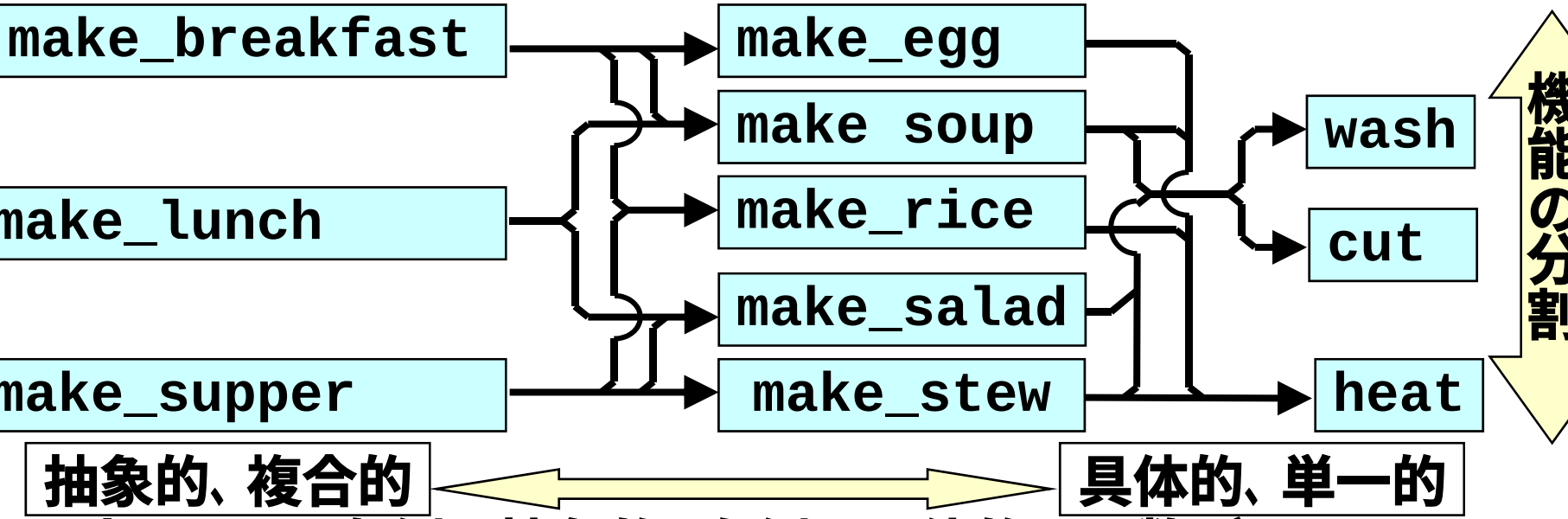
■ご飯、および、サラダを作る処理も共通化したとします。このとき、次のような関係になります。



■これで共通化は終わりでしょうか？材料に対する処理(洗う、切る、加熱する、)も共通化出来ることにしてみましょう。

(6．3) 階層化

■洗う(wash)、切る(cut)、加熱する (heat) も共通化すると、次のようになります (ご飯は無洗米)。

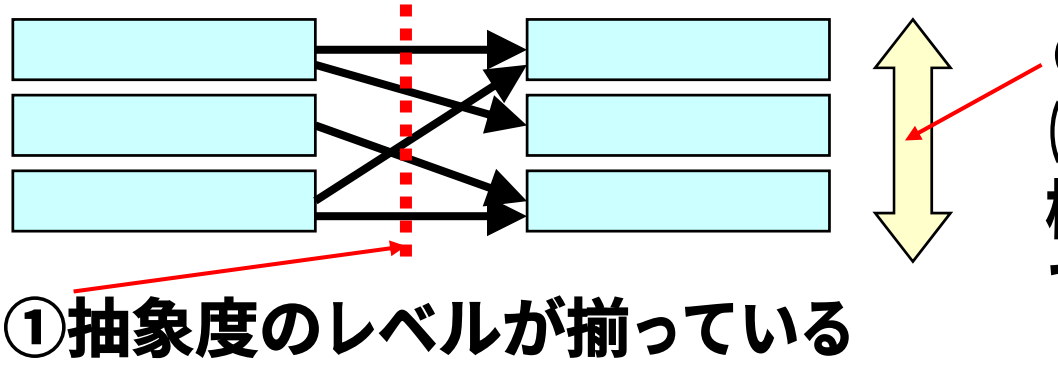


■上図では、左側に抽象的、右側に具体的な関数が配置されています。このような段階が生じることを階層化、段階になっていることを階層構成と言います。また、縦方向には、機能の分割が行われています。

■このような図を、“関数関連図”と呼ぶことがあります。

(6．4) 階層構成

■[意味的な分割]として、良い階層構成を考えて見ます。



- ①例えば、前スライド (6．3) の階層構成の一番右側のレベルは、“洗う”、“切る”、“加熱する”という基本動作として、内容が揃っています。かつ、これらは真ん中のレベルの関数からのみ呼ばれています。
- ②“洗う”、“切る”、“加熱する”の動作には重複がありません。現実の料理方法として充足しているとは言えませんが、ここでは充足していることと理解しておきます。

(6．5) 至るところで。。。

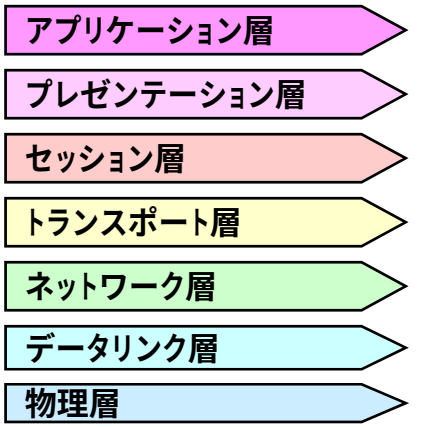
■階層構造は、様々なソフトウェアの中に現れてきます。

■おそらく一番多く現れる階層構成は、次のような階層です。

サービスに近い ⟷ ハードウェアに近い

■例えば、OSIの7層モデルも、
階層構成のひとつです。

■OSIの7層モデルのように決まって
いるものはそれに従う訳ですが、



自ら設計するソフトウェアが階層化される場合、その構成には十分注意を払って設計する必要があります。すなわち、①抽象度のレベルが揃っているか、②重複がなく、充足しているかを考えなければなりません。

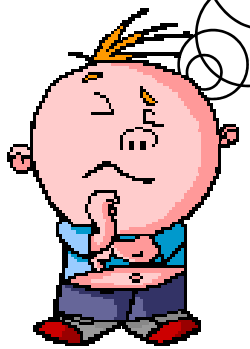
(6．6) オブジェクト指向との関係

- 「オブジェクト指向」と「ソフトウェアの階層構成」との間には、ちょっと微妙な関係があります。
- スライド (6．3) にも示したように、階層構成を取るソフトウェアは、基本的に「機能分割」を行います。
- オブジェクト指向は、「機能分割では駄目だ、“もの”を単位に分割すべきだ」という考えから生まれてきました。
- オブジェクト指向プログラミングでは、「機能分割」をなるべく排除すべきであることは確かですが、それでは「階層構成」的な面が全くないかというと、そうでもありません。クラス間の関係や、メソッド間の関係が階層を成す場合もあります。しかしながら、オブジェクト指向の教科書で、「階層を考えて…」というような記述はまず出てきません。
- 「階層化の考えの素養を持ちながら、それはあまり表に出さないで、オブジェクト指向に相対する」ことが、大人の対応というものではないかと講師は思います。

(6．7．1) 演習 (課題6－1)

■課題6－1：街並みの風景画を作成するプログラムを考えます。構成を考えながら、関数を定義するとともに、関数関連図 (スライド (6.3) 参照) を作成してください。

- 街並みには、民家、商店、学校が描かれます。
- 関数の定義は、その機能が分かるような
- 名前を付けることだけでOKです。
- ヒント：「窓、屋根、壁、大時計、看板」、
  「面、線、円、文字」
- 解釈は、ひととおりではありません。



(6．7．2) 演習 (課題6－2)

■課題6－1で作成した関数関連図に基づき、右のようなプログラムを作成してください。

- 文字列“民家”の出力は、メソッド“drawMinka”で行ってください。“商店”、“学校”の文字列も同様です。
- 文字列“屋根”の出力は、メソッド“addYane”で行ってください。“壁”、“窓”、“看板”、“大時計”の文字列も同様です。
- 文字列“面”の出力は、メソッド“paintMen”で行ってください。“線”、“字”、“円”の文字列も同様です。

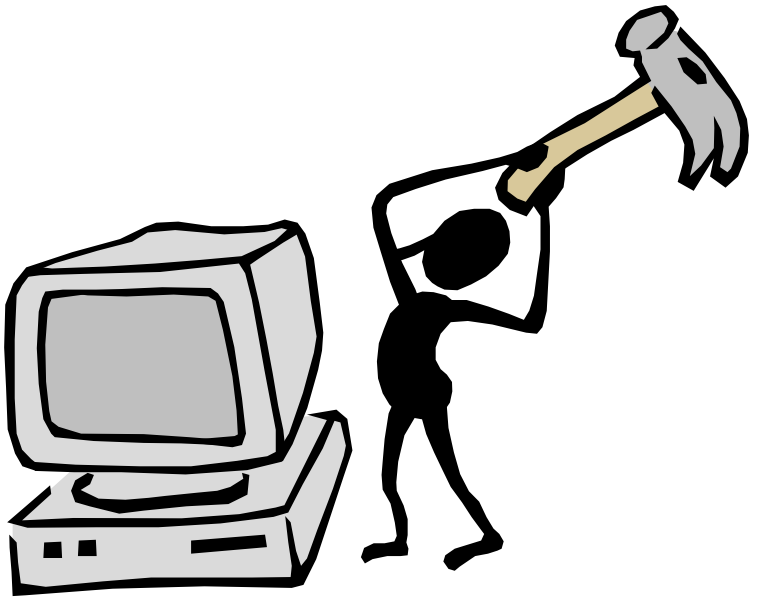
```
c:¥javawork>java Kadai13
民家
  屋根
    面
    線
  壁
    面
    線
  窓
    面
    線
商店
  屋根
    面
    線
  壁
    面
    線
  窓
    面
    線
  看板
    面
    線
    字
```

(以下、学校は省略)

(7) 制御構造を崩すもの

■PADレベルでの設計では、解は一通りとはならないものの、それほど大きな違いは生じないものです。違うものが出来上がっても、素直に考えられたものであれば、それぞれを理解することはあまり難しくありません。

■ところが、“一時変数”の使い方によっては、無意味に処理方法が違ったプログラムが出来てしまいます。これについて見ていきます。



(7．1) 色のプログラム

■次のようなプログラムを考えます。

- 文字列を入力する。
- 文字列に、指定された文字がすべて含まれて居た場合は、これに対応する単語を番号(#)順に出力する。

#	文字	単語
1	r,g	magenta
2	r,b	yellow
3	r,g,b	white

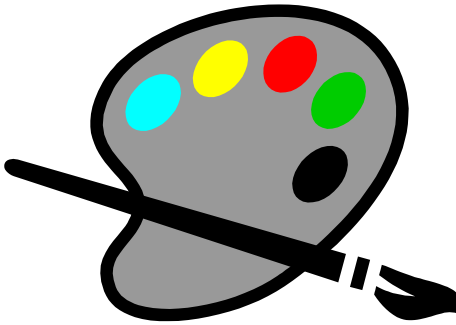
■動作例

```
% kadai10 rgb
magenta
yellow
white
% kadai10 rb
yellow
```

■まず、素直な解を見ていきます。

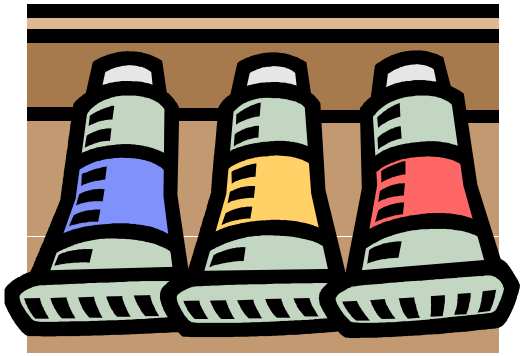
(7．2．1) 素直な解　－その1－

■特に説明の必要のない処理だと思います。



(7．2．2)プログラム　－その1－

```
■import java.io.*;
■public class kadai10{
■  public static void main(String args[]){
■    // 入力
■    String str = args[0];
■    // 含まれる文字を検査する。
■    boolean red, green, blue;
■    if(str.indexOf('r')>=0) red=true; else red=false;
■    if(str.indexOf('g')>=0) green=true; else green = false;
■    if(str.indexOf('b')>=0) blue=true; else blue = false;
■    // 出力を行う。
■    if(red && green) System.out.println("magenta");
■    if(red && blue) System.out.println("yellow");
■    if(red && green && blue) System.out.println("white");
■  }
■}
```



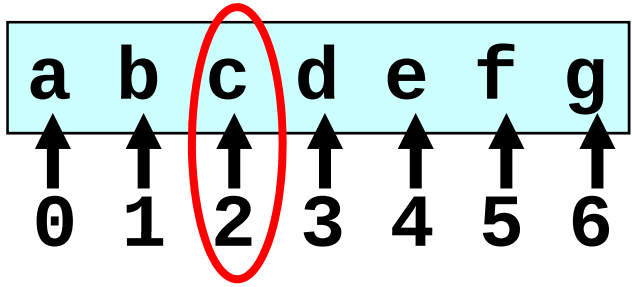
(7．2．3) Stringクラス

■ここでは、色のプログラムに必要な、“indexOf(char)”メソッドの使用例のみを記します。

■右のプログラム(部分)を実行すると、“x=2”が出力されます。

```
1. String str = “abcdefg”;
2. int x = str.indexOf(‘c’);
3. System.out.println(“x=”+x);
```

■すなわち、”indexOf(char)”メソッドは、引数の文字が、Stringクラスの文字列の中で最初に現れる位置を返します(Javaでは何事も0から数えます)。

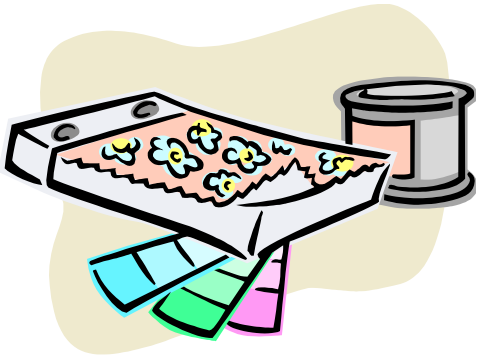


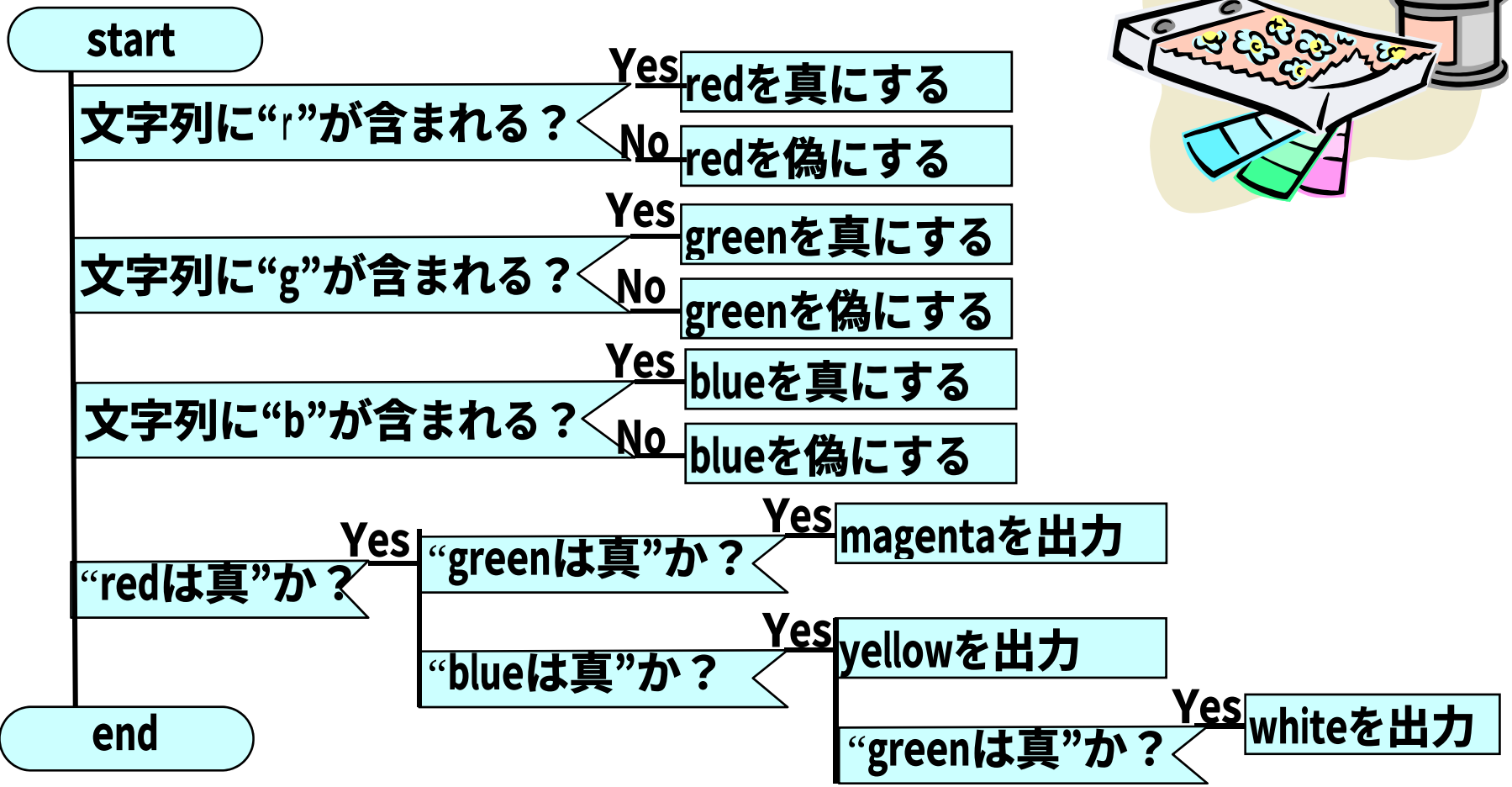
■引数の文字が現れない場合は、”-1”を”indexOf(char)”メソッドは返します。すなわち、右のプログラム(部分)の判定は、「“c”が含まれるか？」を判定します。

```
1. if(str.indexOf(‘c’)>=0){
2.     :
3. }
```

(7．3．1) 素直な解　－その2－

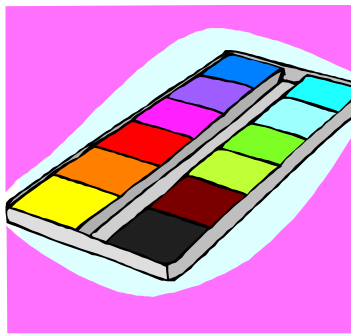
■この解では、処理の流れを考えて、判定を順次行うことにしています。





(7．3．2)プログラム　－その2－

```
■import java.io.*;
■public class kadai10_1{
■  public static void main(String args[]){
■    // 入力
■    String str = args[0];
■    // 含まれる文字を検査する。
■    boolean red, green, blue;
■    if(str.indexOf('r')>=0) red=true; else red=false;
■    if(str.indexOf('g')>=0) green=true; else green = false;
■    if(str.indexOf('b')>=0) blue=true; else blue = false;
■    // 出力を行う。
■    if(red){
■      if(green) System.out.println("magenta");
■      if(blue){
■        System.out.println("yellow");
■        if(green) System.out.println("white");}
■    }
■  }
■}
```



(7．4) 2つの解の比較

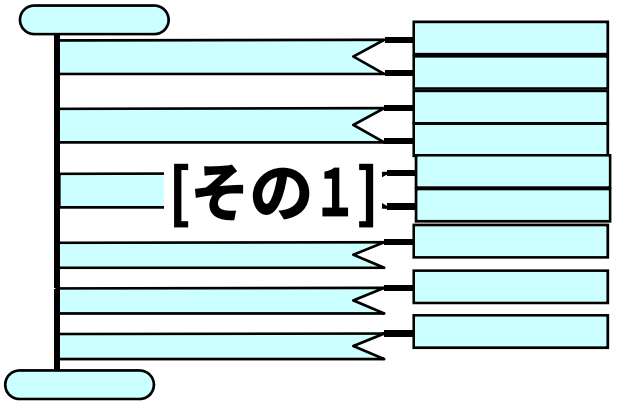
■色のプログラムのような単純な例では、[その1]と[その2]とに、あまり優劣はありません。

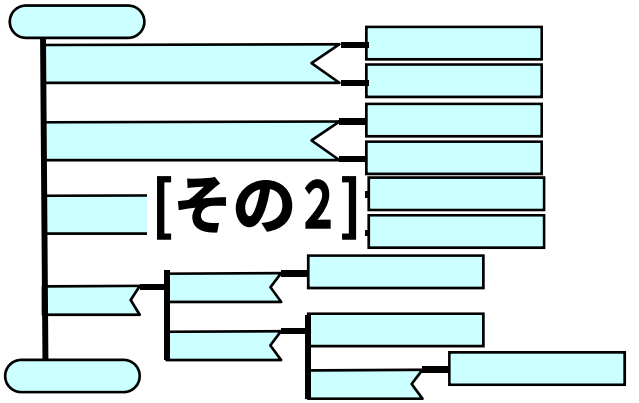
■この場合は、おそらく、[その1]の方が明解であると感じる方が多いと思います。

■しかしながら、「”r”が文字列に含まれていないと、何も出力しない」といったことは、[その2]の方で明解に現れています。

■どちらが優れているかは、プログラムに依る訳で、一般的なことはあまり言えません。

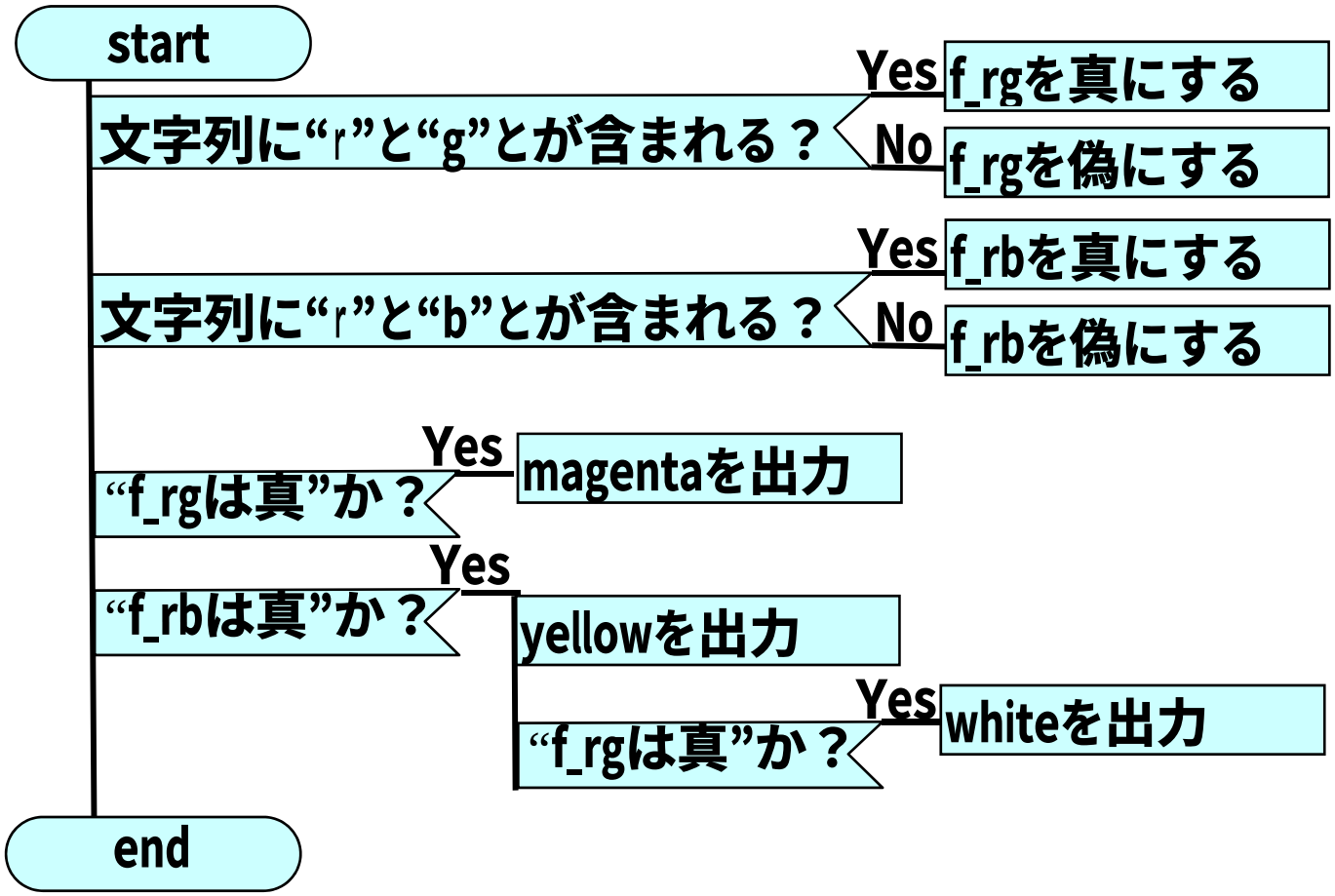
■いずれの方法も、やろうとしていることが素直に現れた形をしています。

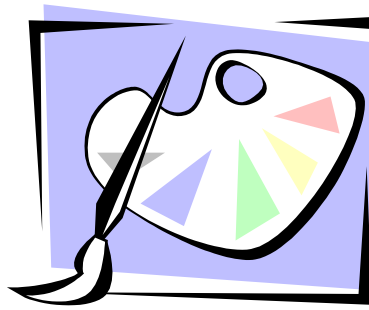




(7．5．1) 素直でない解

■下図の解では、“f_rg”と“f_rb”という名前の一時変数が導入されています。





(7．5．2) プログラム

```
■import java.io.*;
■public class kadai10{
■  public static void main(String args[]){
■    // 入力
■    String str = args[0];
■    // 含まれる文字を検査する。
■    boolean f_rg, f_rb;
■    if(str.indexOf('r')>=0 && str.indexOf('g')>=0)
■     f_rg=true; else f_rg=false;
■    if(str.indexOf('r')>=0 &&str.indexOf('b')>=0)
■     f_rb=true; else f_rb = false;
■    // 出力を行う。
■    if(f_rg) System.out.println("magenta");
■    if(f_rb){
■      System.out.println("yellow");
■      if(f_rg) System.out.println("white");
■    }
■  }
■}
```

(7．5．3) 何が悪いか？

■色のプログラムの例は単純で、スライド(7.5.1)を見て何をしたいかを理解することは難しくありません。それでも、「あれ？」と思われた方もいると思います。

■「あれ？」と思うのは、“white”を出力する際に、次のような条件を用いているからでしょう。

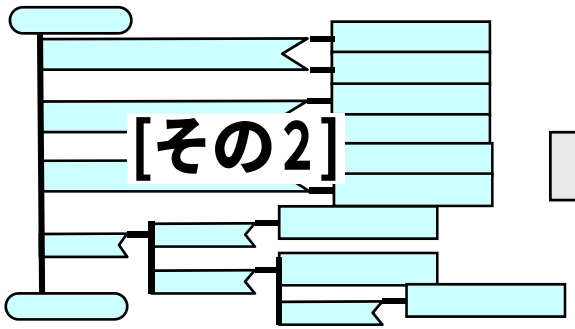
```
1. if(f_rg){
2.     ：
3.   if(f_rb)//whitleを出力
```

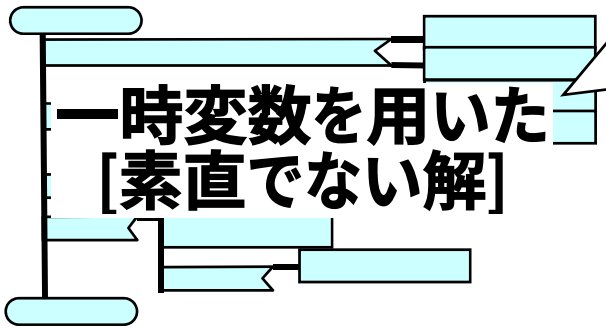
f_rg “r”と“g”とがある かつ f_rb “r”と“b”とがある ⇨ “white”出力 “r”と“g”と“b”とがある

■“f_rg”、“f_rb”のような一時変数を用いることが問題なのです。

(7．6) PADの形

- 繰り返しになりますが、色のプログラムの例では、“f_rg”、“f_rb”は、大して難しい条件で設定されている訳ではありません。
- しかしながら、(7.2)(7.2)で用いた“red”、“green”、“blue”の一時変数に比べると、明確さは劣ります。
- そのような明確さの劣る一時変数を組み合わせて判定条件を作ると、「あれ？」ということになります。
- もっと複雑なプログラムの中で、さらに明確さの欠ける一時変数を用いると、破滅的な事態に陥ることがあります。
- 特殊な一時変数を持ち込むと、PADの形が変わってしまうことに注目してください。





与えられた問題の構造を反映していない形になる！

(7．7) 陥りやすい罠

■色のプログラムで、(7.5)のような「素直でない解」を思い浮かべる人は、まず居ないでしょう。

■比較的長くて複雑なプログラムになると、前のほうの判定結果を覚えておいて、後のほうで使いたくなる場合が起こります。そのようなときに、明確でない一時変数が持ち込まれることがあります。

■一時変数により覚えておくことがすべて悪い訳ではありませんが、面倒くささから「明確な意味」を持たせることが出来ない一時変数を持ち込むと、「あれ？あれれれ！」ということになります。その風味は、“スパゲッティ”に似ていなくもありません。

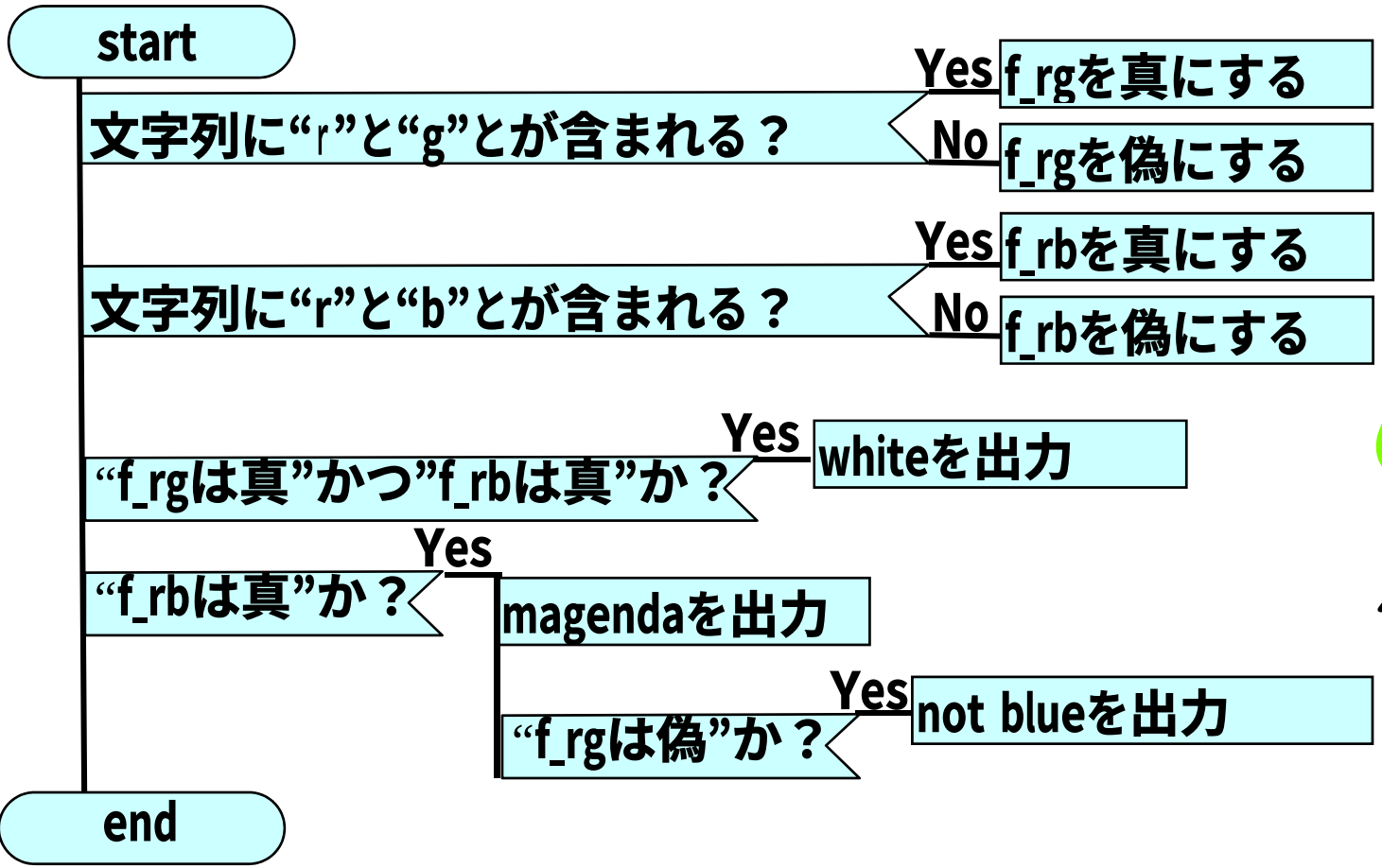
```
1.      :
2.   if(...){
3.       flg=“yes”;
4.   }
5.      :
6.      :
7.      :
8.      :
9.   if(flg &&...){
10.     :
```

①こっちの条件と同じだから、覚えておいて、

②こっちで使ってやろう！

(7．8) 演習(課題7－1)

■次の「素直でないPAD」を、「素直なPAD」に直してください。



(7．9) ヒント：課題７－１

■課題７－１は、次の問題と同じことになります。

- 文字列を入力する。
- 文字列が、指定された文字に関する条件を満たしていれば、これに対応する単語を番号(#)順に出力する。

#	文字に関する条件		単語
	すべて含まれる	含まれて居ない	
1	r,b,g		white
2	r,b		magenta
3	r,b	g	not green

■動作例

```
% kadai11 rgb
white
magenta
% kadai11 rb
magenda
not green
```

おわりに

山のあなたの空遠く
“良いプログラム”住むと人のいふ。
ああ、われビットと尋(と)めゆきて、
涙さしぐみかへりきぬ。
山のあなたになほ遠く
“良いプログラム”住むと人のいふ。

- “良いプログラム”という言葉に、時としてソフトウェア技術者はある種の切なさを感じるのかも知れません。
- 設計はルーチンワークではありません。１００％の知識を持ち合わせていても、“良いプログラム”が設計できるとは限りません。
- 出来るだけのことを学び、人の設計をたくさん見た上で、“真剣に考える”ことに費やした時間だけが、受講者の皆さんの上達を約束するものだと思います。